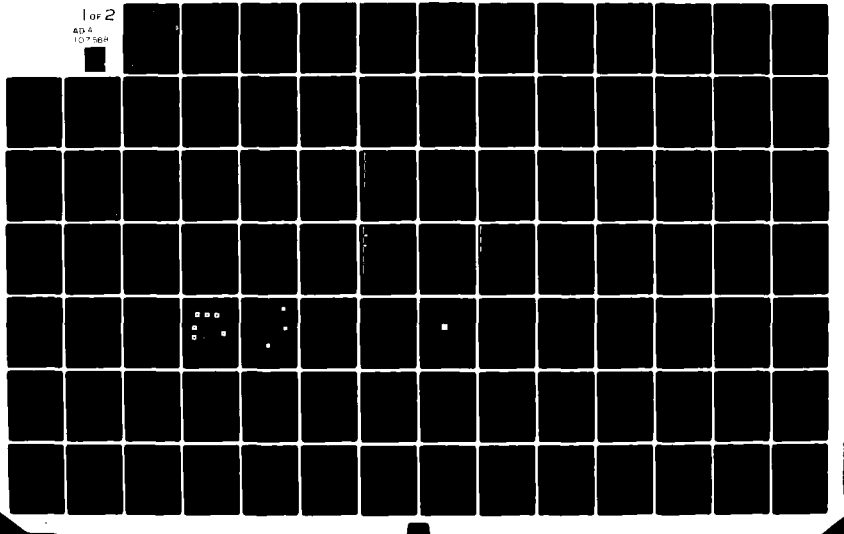
AD-A107 568 STANFORD UNIV CA STANFORD ELECTRONICS LABS F/G 9/2
SYSTEM CONSIDERATIONS IN THE DESIGN OF RESIDUE PROCESSORS.(U)
MAR 79 A HUANG, J MANDEVILLE, J E GOODMAN AFOSR-77-3219
UNCLASSIFIED SU-SEL-79-008 AFOSR-TR-81-0744 NL



LEVEL II

(1)

SEL-79-008

SYSTEM CONSIDERATIONS IN THE DESIGN OF RESIDUE PROCESSORS

Alan Huang
Jon Mandeville
Joseph W. Goodman
Satoshi Ishihara

March 1979

Annual Technical Report No. L722-2

Research sponsored by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under Grant No. AFOSR-77-3219. The United States Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation hereon.

Information Systems Laboratory
Stanford Electronics Laboratories
Stanford University
Stanford, California

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1 REPORT NUMBER: AFOSR-TR- 81-0744	2 GOVT ACCESSION NO.: AD A107568
	3 RECIPIENT'S CATALOG NUMBER
4 TITLE (and Subtitle): System Considerations in the Design of Residue Processors	5 TYPE OF REPORT & PERIOD COVERED: Annual Report 2.1.78.-1.31.79.
	6 PERFORMING ORG. REPORT NUMBER: SEL-79-008
7 AUTHOR(s): Alan Huang, Jon Mandeville, Joseph W. Goodman, S. Ishihara	8 CONTRACT OR GRANT NUMBER(s): AFOSR-77-3219
9 PERFORMING ORGANIZATION NAME AND ADDRESS: Stanford University, Stanford, California, 94305	10 PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F 2305/B1
11 CONTROLLING OFFICE NAME AND ADDRESS: United States Air Force, Air Force Office of Scientific Research, Bldg. 410, Bolling AFB, D.C. 20332	12 REPORT DATE: 3.31.79. / 13 NO. OF PAGES: 125
14 MONITORING AGENCY NAME & ADDRESS (if diff. from Controlling Office)	15 SECURITY CLASS. (of this report): Unclassified
	15a DECLASSIFICATION/DOWNGRADING SCHEDULE

16 DISTRIBUTION STATEMENT (of this report)

Approved for public release; distribution unlimited.

17 DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from report)

18 SUPPLEMENTARY NOTES

19 KEY WORDS (Continue on reverse side if necessary and identify by block number)

Residue arithmetic
Optical data Processing
Integrated Optics
Optical Computing

20 ABSTRACT (Continue on reverse side if necessary and identify by block number)

The possible structure of a residue computer based on read-only-memories (ROMs) is considered. Methods for implementation with either electronic or optical ROMs is discussed. Methods for reducing errors in residue computations are discussed.

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

I. INTRODUCTION

This report describes the work accomplished under grant AFOSR-77-3219 from the Air Force Office of Scientific Research during the period 1 February 1978 through 31 January 1979.

The work is divided into two separate and distinct parts. Section II, written by Alan Huang, is concerned with a variety of architectural issues in the design of any computer based on residue arithmetic. The results are directly applicable to optical approaches to residue computing, but are equally important in considering the design of electronic versions of this type of computer.

Section III, written by Jon Mandeville, considers the problem of error detection and correction in residue computers. As interest grows in the use of residue techniques in what are basically analog systems, the consequences of errors in the representation of residue numbers by analog values must be considered. The issue is a complex one, for often the probability of error will increase as the size of a given modulus increases, so the choice of moduli will affect the reliability of the system. Sometimes extra moduli can be used for the sole purpose of protecting the integrity of the data. These issues are explored in detail in section III.

Our earlier report [Ref. 1] is essential background reading before begining this report. A sightly condensed version of the earlier report can be found in the January 15th issue of Applied Optics.

II. ARCHITECTURAL STUDIES

A. Background

The combination of residue arithmetic and optics was prompted by the realization that the cyclic nature of the residue number system can be mimicked by various physical phenomena. Our initial investigation resulted in a genealogy of possible approaches as shown in Table 1. In an effort to evaluate the relative advantages of the various approaches we studied the physical switching mechanisms ultilized by each approach. Our findings are summarized in Table 2. Speed, possibility of integration, and cost were used as some of the criteria. These results were then incorporated into an overall evaluation of the basic approaches, as summarized in Table 3 *.

The approaches listed in the preceeding table involve a direct mimicking of the residue system. Some mathematical insights have enabled us to relax some of the restraints on these technologies. One simplification was the realization that any cyclic shift can be synthesized from a binary decompostion of shifts. As an example, a cyclic shift of 7 is equal to a shift of 4, a shift of 2, and a shift of 1. This result reduced the type and number of shifts needed.

Another important finding was that the non cyclic maps needed for multiplication can be generated with a fixed pre-permutation, cyclic shifts, and a post-permutation [Ref. 1, p. 36]. This allows technology that was previously only capable of performing cyclic shifts to also

* Tables 1, 2, and 3 prepared by Dr. Yoshito Tsunoda of Hitachi Ltd. during his stay at Stanford from 1976 thru 1977.

TABLE 1

OPTICAL IMPLEMENTATION OF RESIDUE ARITHMETIC UNIT

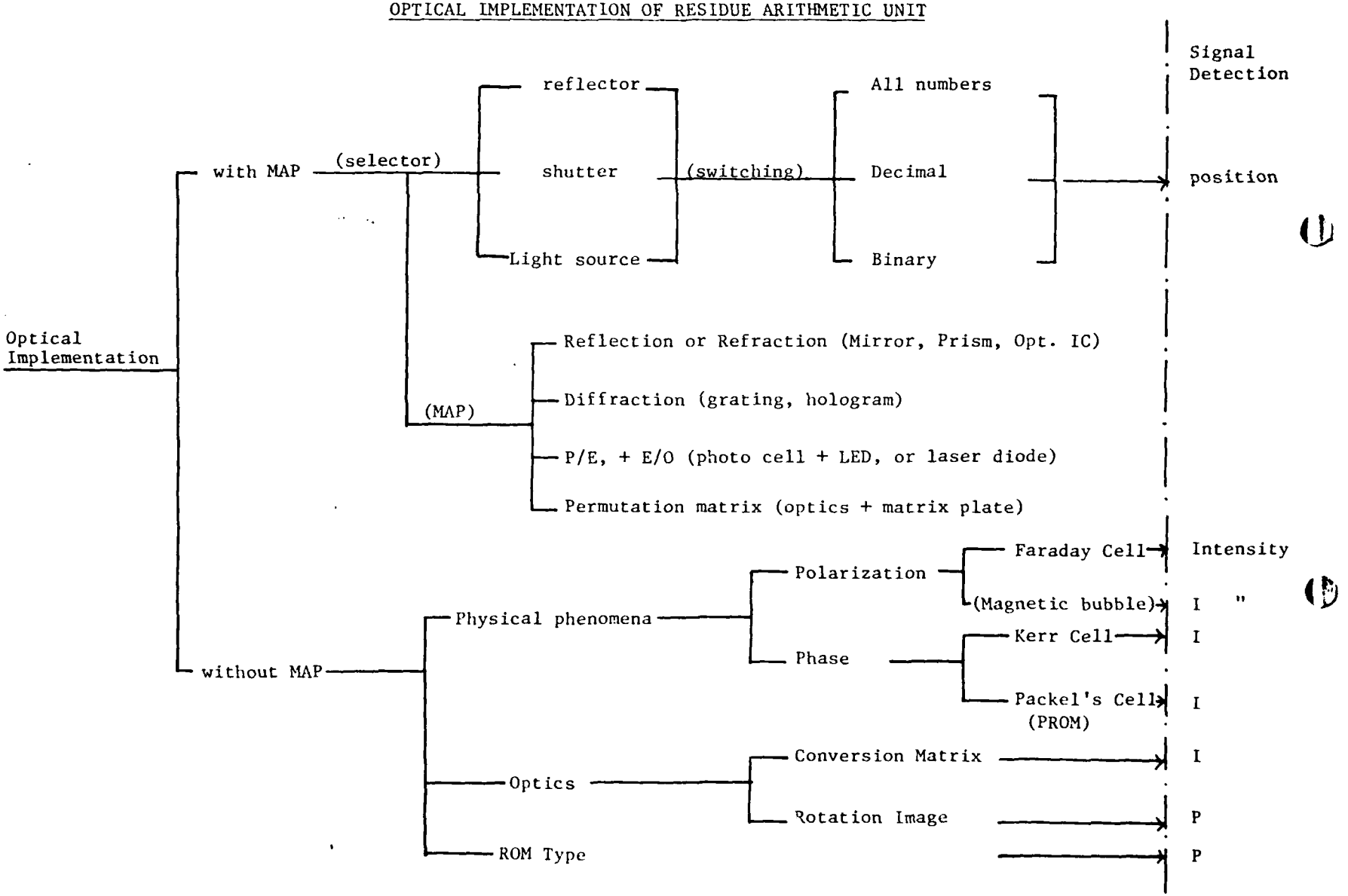


TABLE 2

CHARACTERISTICS OF SEVERAL SWITCHING DEVICES

	Switching Speed	Working Voltage	Life-time	Possibility of Integration	Price	Volume	Evaluation
A/O Deflector	1 μsec~20 nsec	~100	Good	Fair	Expensive	Large	Good
Faraday Cell	~1 μsec		Good	Bad	Expensive	Large	
Kerr Cell (KTN)	~30 nsec	~300	Good	Bad	Expensive	Large	
Laser diode	~1 nsec	~1	Fair	Good	Expensive→ Inexpensive	Small	Good
LED	~1 nsec	~1	Good	Good	Inexpensive	Small	Good
E/O Modulator (KDP)	~10 nsec	~K	Good	Bad	Expensive	Large	
Ferroelectric Crystal	100 ~ 1 μsec	100~200	Good	Fair	Expensive	Small	
PLZT Ceramics	10~1 μsec	50~200	Fair	Fair	Inexpensive	Small	
Liquid Crystal	1 sec(scatter) 30 msec (bire-fringence)	~10	Good	Bad	Inexpensive	Small	
Membrane	~100 μsec	100	Good	Good	Inexpensive	Small	
Titus	30 msec	100	Good	Bad	Expensive	Large	

perform multiplication. These technologies could then perform decoding to mixed radix as well as other more complex computations.

These findings formed a foundation for a second generation of implementations. Our efforts had been directed at mimicking cyclic shifts with physical cyclic shifts. Such shifts can also be accomplished by several indirect means.

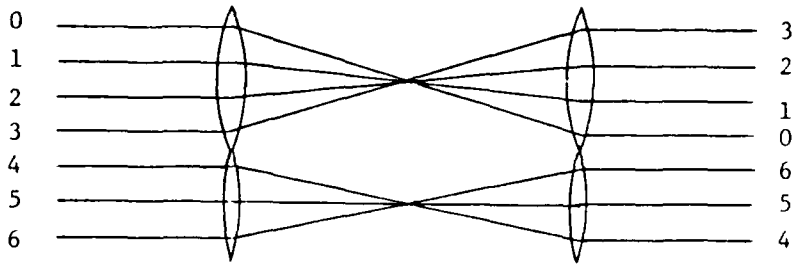
The first such approach was the off diagonal switching device [Ref. 1, p. 34]. Mathematically, this approach relies on the fact that the partitioning of an ordered set, the reversal of the elements in each of the two subsets, and the reconcatenation of the two sets results in a cyclic shift in reverse order. As an example {0,1,2,3,4,5,6} would be partition into {0,1,2,3} and {4,5,6}. Each of these subsets would then be reversed. The result is {3,2,1,0} and {6,5,4}. Recombining these two subsets would result in {3,2,1,0,6,5,4} which can be seen to be the reverse of {4,5,6,0,1,2,3} which is a cyclic shift of 3. The primative operations in this type of cyclic shifter are a partition , reversal, and concatenation. This finding extends the types of technologies that are capable of performing cyclic shifts. A simple conceptual example is shown in Fig. 1(a), in which two lenses are used to perform a cyclic spatial shift.

Another second generation mapping device is the "amida kuzi" approach [Ref. 1, p. 41]. The mathematical basis of this approach is that any permutation (1-to-1 mapping) can be accomplished with only interchanges between neighboring elements. As an example, a cyclic shift of 3 of the set {0,1,2,3,4,5,6} can be accomplished by modifying the order with neighboring interchanges to {0,1,2,4,3,5,6} to

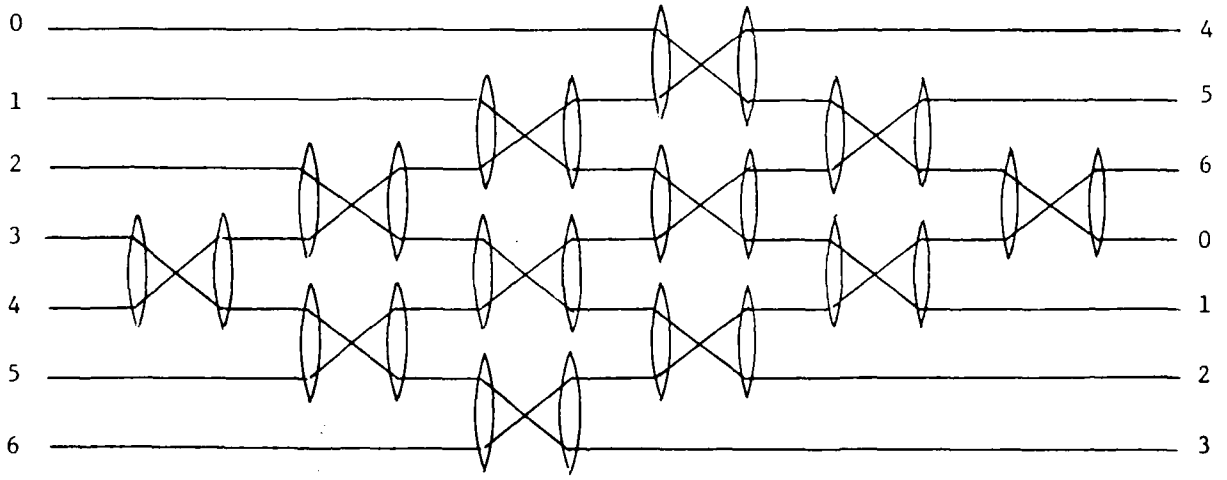
TABLE 3

OVERALL EVALUATION OF VARIOUS IMPLEMENTATIONS

	Key device	Price	Speed	Multiplex	Input No.	Special Function	Optical Efficiency	Levels to be Detected
MAP Mirror or Prism	MAP Deflector	Low High	≈20 μsec Mid	Good Bad	Large	Good	Good	1
LED	MAP	Low	≈1 μsec High					
Polarization	Faraday Cell	High	Low	Bad	Small	Bad	Good	30
Phase	Kerr Cell Pockel's Cell	High	High	Bad	Small	Bad	Good	30
Permutation matrix	Mask} Lens} Shutter	Low High	Mid	Good	Mid	Good	Bad	1
Conversion matrix	Mask} Lens}	Low	High	Good	Large	Bad	Fair	~20



(a) CYCLIC SHIFTS WITH REVERSALS



(b) CYCLIC SHIFTS WITH INTERCHANGES

FIG. 1: SECOND GENERATION SHIFTERS

{0,1,4,2,5,3,6} to {0,4,1,5,2,6,3} to {4,0,5,1,6,2,3} to {4,5,0,6,1,2,3} to finally {4,5,6,0,1,2,3}, which is a cyclic shift of 3. The primative operation of this type of cyclic shifter is a pairwise interchange. This finding also extends the types of technologies that are capable of performing cyclic shifts. A simple conceptual example is shown in Fig. 1(b), where couplets of lenses are used to perform interchanges for a cyclic permutation.

These second generation approaches to performing permutations extend the range of technologies to be considered. This varity makes it more difficult to focus on any "best" approach.

In an effort to gain a better perspective, this year's efforts were devoted to examining both the potiental and limitiations of using various technologies. Phrased in another way "given a certain technology what can and cannot be done with it?"

B. Given A Certain Technology What Can We Do With It?

The potential of such systems is dependent on the types of problems that it can solve and the throughput (data samples/sec) with which it can solve these problems.

Some of the problems that seem attractive for a residue approach are inner products, summation, and determinate evaluation. They are representative of many of the problems in signal processing. They also provide a convienent benchmark for comparisons with more conventional computational approaches.

A residue approach is capable of very large throughputs. There are two fundamental ways of achieving this goal. One appoach relies on speed while the other relies on parallelism. These two architectural strategies favor different technological traits. The overall goal is to relate the characteristics of a given technology to its performance on a given problem using a given architectural approach. This will help to define the appropriateness of a given technology and also to quantify the relative merits of the a residue and conventional computational approaches.

C. Modular Processors Pipelined By Moduli (The Rabbit)

A modular processor pipelined by moduli is shown in figure 2. This architectural approach relies on ultilizing speed to achieve high throughputs. We refer to this approach as the "rabbit" approach or as a cascaded number theoretic processor. The processor works in much the same way as an assembly line. The coefficients of a desired calculation are placed on the data bus at the left. The MOD X processor takes these coefficients and produces the modulus X equivalent of the answer for the desired calculation. This answer is also one of the mixed radix coefficients of the answer; it is passed, along with the coefficients of the calculation, by the data bus to the MOD Y processor. This processor uses the coefficients along with the previously determined mixed radix coefficients to compute another mixed radix coefficient. Meanwhile the MOD X processor is processing the coefficients of another calculation. This process continues. Each modular processor uses the coefficients along with all the previously determined mixed radix coefficients to produce another mixed radix coefficient. What emerges from the data bus is a mixed radix version of the answer to the desired calculation. The throughput rate of this pipelined processor is the reciprocal of the maximum modular processor time. The last processor usually takes the longest since it has to consider the results of all the previous processors. The latency, the time it takes for a given calculation to complete this assembly line, is the product of the number of processors and the maximum processor time. The accuracy or range of such a processor depends only on the product of the moduli used. If a processor is designed with many small moduli rather than a few large moduli then the latency will be larger because there would be more modular processors.

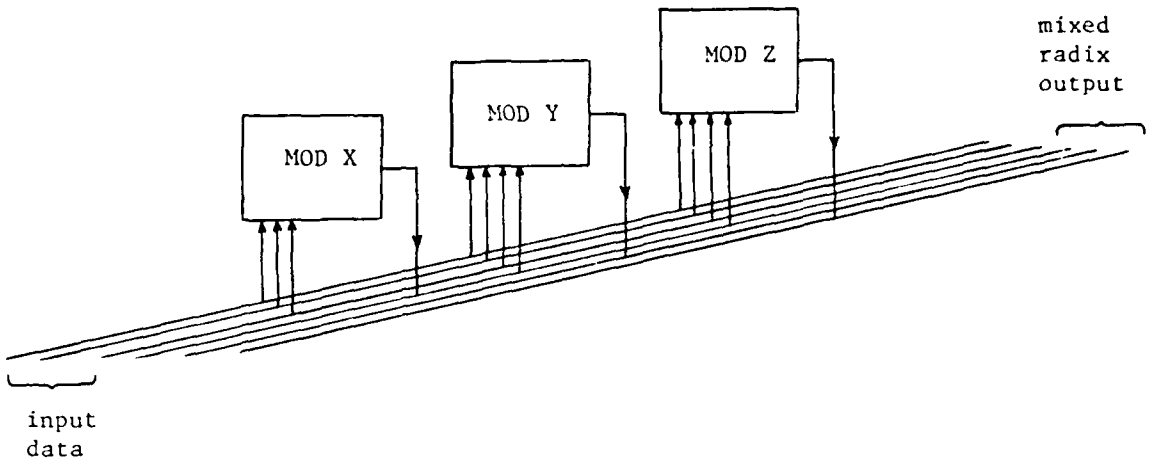


FIG. 2: PIPELINING BY MODULI

The throughput rate of this process is independent of the accuracy. This is different from conventional computational approaches.

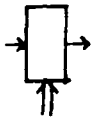
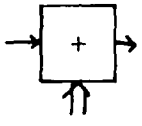
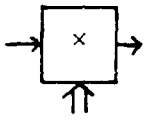
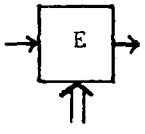
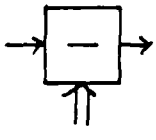
The throughput of this type of system is dependent on the maximum modular processor time. For analytical purposes it was useful to establish some basic building blocks for modular processors. By studying the behavior of these simpler structures the behavior of the modular processors can be synthesized and the overall performance of such systems predicted. The basic building blocks are shown in Fig. 3. The operation of each of the building blocks for a given technology can be characterized by two parameters. One is the set time; this is how long the unit needs to get ready. The other parameter is the propagation time; this is the time needed for a signal to propagate through the device once it has been set.

The first unit is a map , i.e. a fixed permutation that needs no set time. The time needed to propagate through the map is denoted as t_p.

The next unit is a permutation primative. It either permutes the incoming signal or bypasses it. It can be viewed as a switching mechanism that directs the signal to one of two fixed maps. The switching mechanism has a set time associated with it, which is denoted as t_s . The signal has to then propagate through the switching mechanism as well as through one of the two maps. This time is denoted as t_{pp}.

The rest of the units can be constructed from these two basic devices and thus have set and propagation times that can be easily derived.

The function of a residue adder is to perform modular addition.

UNIT	NAME	SET TIME	PROPAGATION TIME
	MAP	0	t_p
	PERMUTATION PRIMITIVE	t_s^*	t_{pp}
	RESIDUE ADDER	$t_s + t_m^*$	$\lceil \log_2 m_i \rceil t_{pp}$
	RESIDUE MULTIPLIER	$t_s + t_m^*$	$2t_p + \lceil \log_2 m_i \rceil t_{pp} + t_{pp}$
	RESIDUE ENCODER	t_s^*	$\lceil \log_2 N \rceil t_{pp}$
	RESIDUE SUBTRACTOR	$t_s + t_m^*$	$\lceil \log_2 m_i \rceil t_{pp}$

*If the set signal is optical then an additional detection time of t_d seconds is needed.

FIG. 3: BUILDING BLOCKS

It's structure is shown in Fig. 4. It consists of a cascade of permutation primatives that are controlled by an action table implemented with conventional electronic logic. The action table for a modulus 7 adder is shown in Fig. 5 and Table 4. The columns of the table represent the maps of the various permutation primatives. The rows represent the number to be added. If the number to be added is 5 then that row of the action table indicates that the permutation $P(X) = X + 4$ and the permutation $P(X) = X + 1$ must be activated. Since the permutation primatives are cascaded this will be equivalent to a permutation of $P(X) = X + 5$. Such an action table is a simple table lookup and can be implemented with one level of logic. The set time would thus be the time for one level of logic, t_l, and the set time of the permutation primatives, t_s. The propagation time depends on how many permutation primatives are cascaded, which in turn depends on the number of binary bits needed to represent the modulus. The propagation time would be $\lceil \log_2 m_i \rceil t_{pp}$ where m_i is the modulus and the half brackets indicate the next larger integer.

The structure of a residue subtractor is identical to that of a residue adder, except that the action table is modified. Such an action table for modulus 7 subtraction is shown in Table 5. Modular subtraction is equivalent to addition of a modular complement. To subtract a value of 5 , the permutation $P(X) = X + 2$ must be activated, since 2 is the modular complement of 5 for modulus 7. The set and propagate times are identical to those of a residue adder.

The structure of a residue multiplier is shown in Fig. 6. It consists of two fixed maps, several permutation primatives, and an action

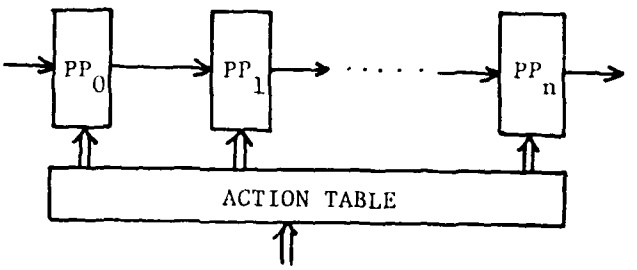


FIG. 4: RESIDUE ADDER

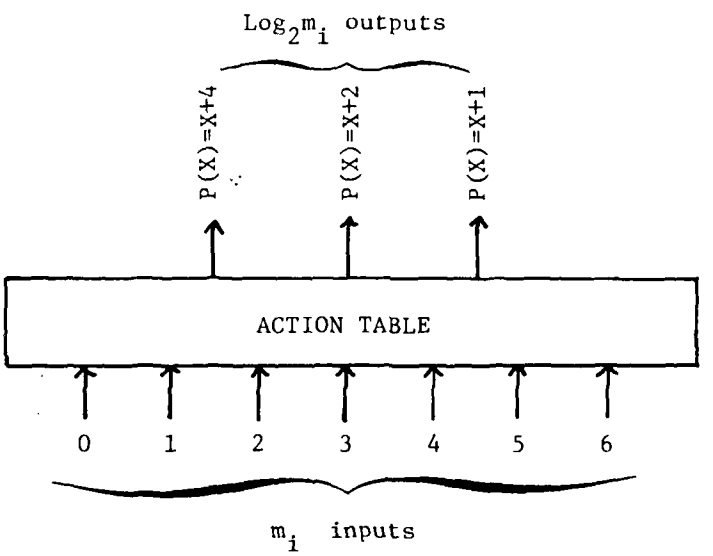


FIG. 5: ACTION TABLE FOR RESIDUE ADDER

+	P(X)=X+4	P(X)=X+2	P(X)=X+1
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0

TABLE 4: ACTION TABLE FOR MODULUS 7 ADDITION

-	P(X)=X+4	P(X)=X+2	P(X)=X+1
0	0	0	0
1	1	1	0
2	1	0	1
3	1	0	0
4	0	1	1
5	0	1	0
6	0	0	1

TABLE 5: ACTION TABLE FOR MODULUS 7 SUBTRACTION

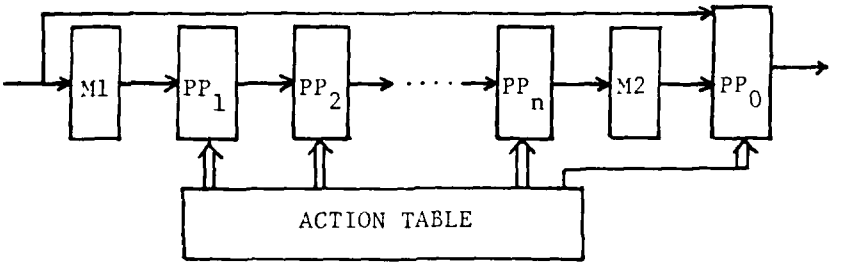


FIG. 6: RESIDUE MULTIPLIER

X	$PP_0(X)=X+0$	$PP_1(X)=X+1$	$PP_2(X)=X+2$	$P(X)=0$
0	0	0	0	1
1	1	0	0	0
2	0	1	0	0
3	0	1	1	0
4	0	0	1	0

TABLE 6: ACTION TABLE FOR MODULUS 5 MULTIPLIER

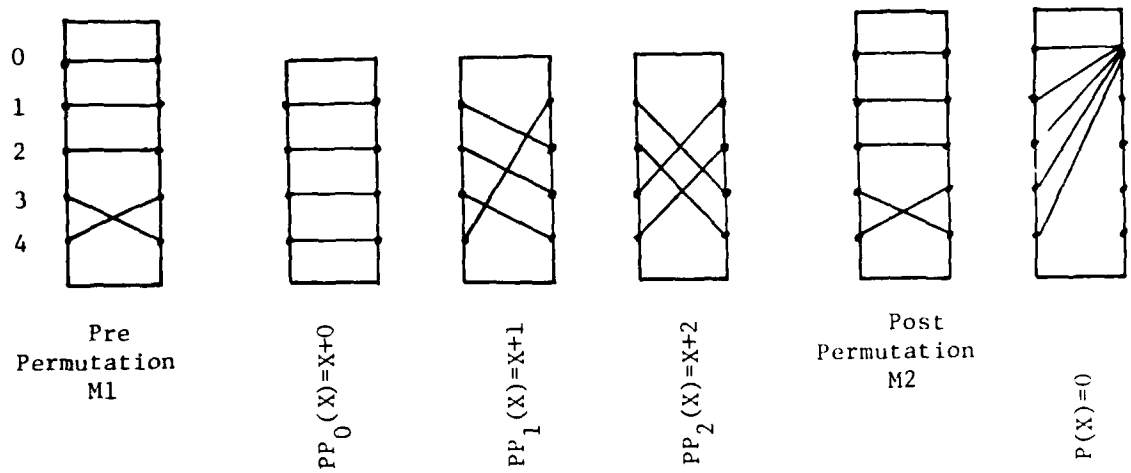


FIG. 7: PERMUTATIONS FOR
MODULUS 5 MULTIPLIER

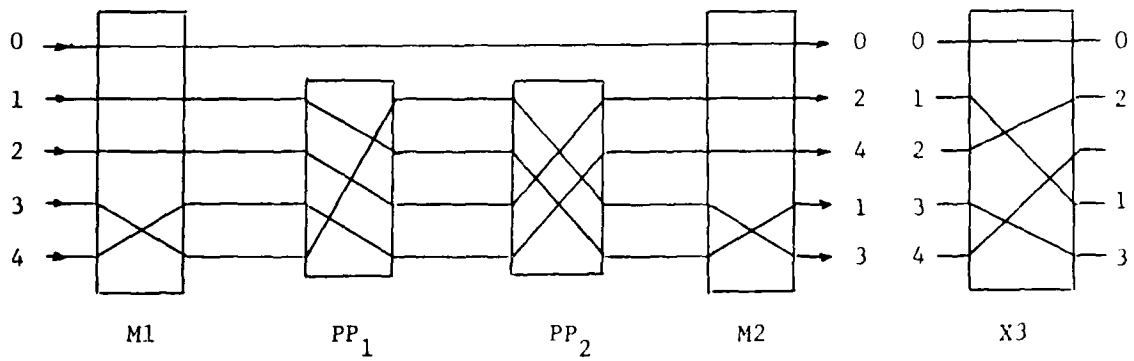


FIG. 8: EXAMPLE OF MODULUS 5 MULTIPLICATION

table. The fixed maps perform the equivalents of modular logs and anti-logs [Ref. 1, p. 36] *. The right-most permutation primative performs the permutation P(X) = 0 which covers the case of multiplication by zero. The other permutation primatives provide various cyclic shifts. The permutations needed for a modulus 5 multiplier are shown in Fig. 7. The action table is shown in Table 6. To multiply by 3 the action table directs permutation primatives PP_1 and PP_2 in Fig. 8 to be activated. If a signal propagates through map M1, permutation primatives PP_1 and PP_2, and map M2, then the permutation will be equivalent to the desired permutation of P(X) = 3X. The set time of such a unit would require a time of t_1 for the action table and a time of t_s to set the required permutation primatives. The propagation time varies with the size of the moduli since this influences the number of permutation primatives needed to provide the required shifts. The propagation time would be $2t_p + \lceil \log_2 (m_i-1) \rceil t_{pp} + t_{pp}$.

The structure of a binary to residue encoder is shown in Fig. 9. It consists of a cascade of permutation primatives. If the number to be encoded is in the form $a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_1 2 + a_0$, then the permutation of the permuation primative PP_n is $P(X) = X + 2^n$. The various bits of the number to be encoded are used to activate the associated permutation primative. Since all the permutation primatives are set in parallel, the set time is just t_s. The propagation time depends on how many permutators are cascaded which in turn depends on the number of bits in the number to be encoded. The propagation time is thus $\lceil \log_2 N_{max} \rceil t_{pp}$, where N_{max} is the largest number to be encoded.

* Also see pages 118-121 of Residue Arithmetic and Its Applications to Computer Technology by N. S. Szabo and R. I. Tanaka, McGraw-Hill 1967

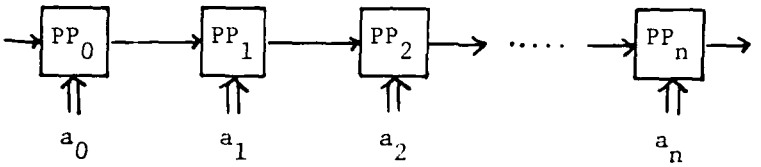


FIG. 9: BINARY TO RESIDUE ENCODER

	P(X)=X+1	P(X)=X+2	P(X)=X+4
0000	0	0	0
0001	1	0	0
0010	0	1	0
0011	1	1	0
0100	0	0	1
0101	1	0	1
0110	0	1	1
0111	0	0	0
1000	1	0	0
1001	0	1	0
1010	1	1	0
1011	0	0	1
1100	1	0	1
1101	0	1	1
1110	0	0	0
1111	1	0	0

TABLE 7: MODIFIED ADDER ACTION TABLE FOR ENCODING

Encoding can also be accomplished by modifying the action table of a residue adder. A modified action table to convert a 4 bit binary number into its modulus 7 equivalent is shown in Table 7. As an example, to encode 1010, which is 10 in binary, the permutations $P(X) = X + 2$ and $P(X) = X + 1$ have to be activated. This produces a permutation of $P(X) = X + 3$. This is correct since 1010 has a net contribution of 3 for modulus 7.

By properly incorporating the effects of the weighting factors associated with the various digits of a number, an action table can be implemented to encode any n bits of a number. Several of these units can be cascaded to encode all the portions of a number. The set time is the same as that of a residue adder. The propagation time would depend on how many such units were cascaded.

It has been assummed in the previous discussion of building blocks that the control signals for the building blocks were electrical. In some of the subsequent discussion the control signal will be optical. An additional detection time of t_d seconds would have to be added to the set times of the devices to represent the time needed to convert from an optical to an electrical signal.

These building blocks can be assembled to form modular processors capable of solving certain types of problems.

(1) Summation Processor

The structure of a modular processor that performs summation is shown in Fig. 10. The coefficients of the desired summation are tapped

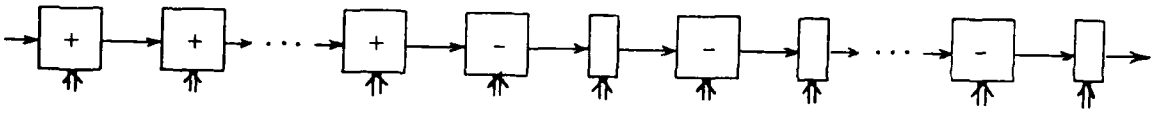


FIG. 10: SUMMATION PROCESSOR

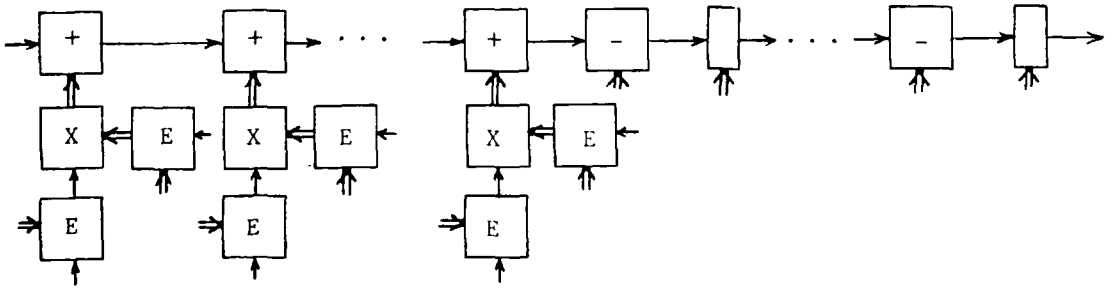


FIG. 11: INNER PRODUCT PROCESSOR

off the bus and used to set the encoders. The subtractors and permutation primatives form the mixed radix decoding section *. They use the previously determined mixed radix coefficients to convert the present result into another mixed radix coefficient. All the building blocks can be set at the same time. The pipelined structure shown in Fig. 2 insures that all the coefficients and previously derived mixed radix coefficients are available to each modular processor. The maximum processor time for a modular summation processor would be

$$t_{max} = (t_s + t_d) + N_d\lceil \log_2 N_{max} \rceil t_{pp} + (N_m - 1)(\lceil \log_2 m_i \rceil t_{pp} + t_{pp}) + t_d$$

$$= (\text{ set encoders, subtractors, and permutators}) +$$

$$(\text{ propagate through encoders, subtractors, and permutators }) +$$

$$(\text{ detect }),$$

where N_d is the number of points to be summed and N_m is the number of moduli used. The throughput rate of such a pipelined system would be the reciprocal of t_{max}. This equation is significant in that it connects the characteristics of a technology with the performance of a system designed to perform a particular type of computation. This connection will aid in evaluating the appropriateness of a particular technology for a particular type of problem.

(2) Inner Product Processor

The structure for a modular processor that performs inner products is shown in Fig. 11. The coefficients of the desired inner product are

* The permutation primatives can be replaced with fixed maps if the error correcting feature described below is not desired. To correct an error, the suspicious result from a previous modular processor is excluded from the decoding process by instructing the associated subtractor to subtract a 0 and the associated premutation primative to bypass, rather than multiply, the result by a constant. This operation would perserve the correctness of a particular calculation at the expense of a reduced range.

tapped off the data bus and used to set all the encoders. The previously derived mixed radix coefficients on the bus are used to set the subtractors and permutation primatives of the decoding section. Signals propagate through all the encoders. The signals of one vector are then used to set the multipliers. The signals associated with the other vector then propagate through these multipliers. These signals then set all the adders. A signal then propagates through all the adders and the decoding section. The maximum processor time for a modular processor that performs inner products would be

$$t_{max} = t_s + \lceil \log_2 N_{max} \rceil t_{pp} + (t_s + t_l) + (2t_p + \lceil \log_2 m_i \rceil t_{pp} + t_{pp})$$
$$+ (t_s + t_l) + N_d \lceil \log_2 m_i \rceil t_{pp} + (N_m - 1)(\lceil \log_2 m_i \rceil t_{pp} + t_{pp}) + t_d$$
$$= (\text{set encoders}) + (\text{propagate through encoders}) +$$
$$(\text{set multipliers}) + (\text{propagate through multipliers}) +$$
$$(\text{set adders and subtractors}) +$$
$$(\text{propagate through adders and subtractors}) + (\text{detect}).$$

If it is assummed that t_{pp} is much smaller than t_s or t_d, then the length of the inner product is not a significant factor. The processor time would then be dominated by the three set times.

(3) Determinant Processor

The evaluation of determinants are useful in the computation of matrix inverses. They are the sums and differences of multiple products. The structure of a modular processor to evaluate determinates is shown in Fig. 12. The coefficients of the desired determinant are tapped off the data bus and used to set all the encoders. The previously derived mixed radix coefficients are used to set the decoding section. Signals

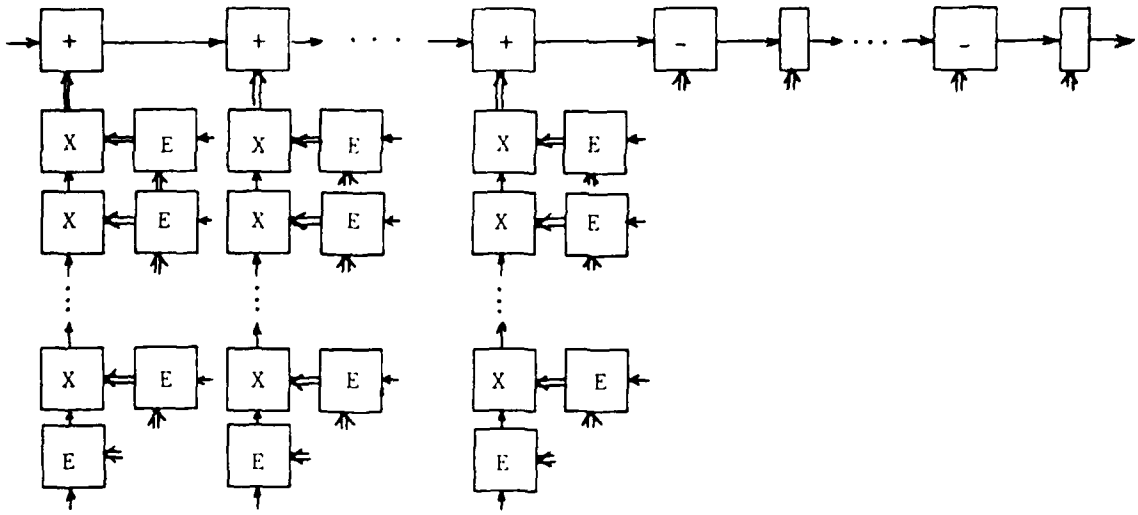


FIG. 12: DETERMINATE PROCESSOR

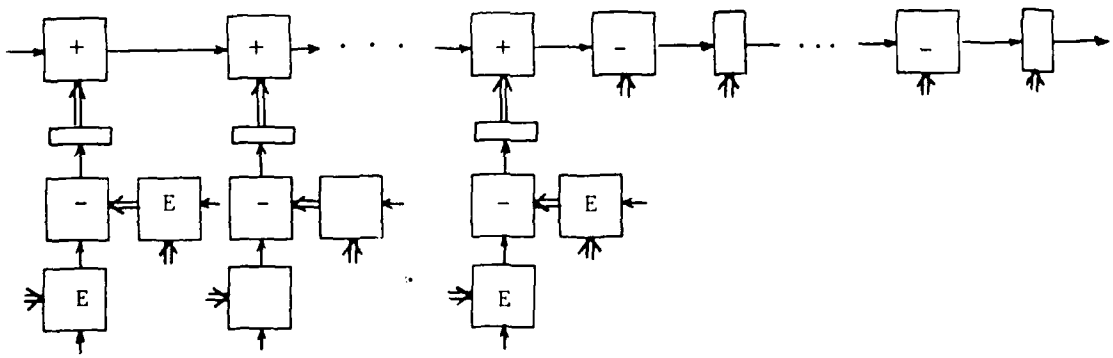


FIG. 13: SQUARED DISTANCE PROCESSOR

propagate through all the encoders in parallel. These signals are used to set the multipliers. Signals propagate through all the multipliers. These signals are used to set all the adders. A signal then propagates through all the adders and the decoding section. The maximum processor time for a modular processor that evaluates determinants would be

$$t_{max} = t_s + \lceil \log_2 N_{max} \rceil t_{pp} + (t_s + t_l) + N_d(2t_p + \lceil \log_2 m_i \rceil t_{pp} + t_{pp})$$
$$+ (t_s + t_l) + N_d \lceil \log_2 m_i \rceil t_{pp} + (N_m - 1)(\lceil \log_2 m_i \rceil t_{pp} + t_{pp}) + t_d$$
$$= (\text{set encoders}) + (\text{propagate through encoders}) +$$
$$(\text{set multipliers}) + (\text{propagate through multipliers}) +$$
$$(\text{set adders and subtractors}) +$$
$$(\text{propagate through adders and subtractors}) + (\text{detect}),$$

where N_d is the dimension of the determinate.

This processor time is approximately equal to that of the inner product processor. It is also dominated by three set times. The only difference is that in this processor, a signal has to propagate through several multipliers rather than just one as in the case of inner products. The larger the matrix involved the more multipliers have to be cascaded. If the propagation time, t_{pp}, is assumed to be much smaller than that of the set time, t_s, it can be seen that the size of the matrix does not strongly influence the processor time.

(4) Squared Vector Distance Processor

The sum of the squares of the differences of two vector components is used as a distance evaluator in many optimizing or decision algorithms. A modular processor to compute such squared distance is shown in Fig 13. The unique feature is that the signals propagate through a

fixed map before they are added. This map performs the polynomial transform $P(X) = X^2$. This example is used to demonstrate that any polynomial with integer coefficient and exponents can be used in these processors. The maximum processor time for a modular processor that computes such distances would be

$$t_{max} = t_s + \lceil \log_2 N_{max} \rceil t_{pp} + (t_s + t_1) + \lceil \log_2 m_i \rceil t_{pp} +$$
$$+ (t_s + t_1) + N_d \lceil \log_2 m_i \rceil t_{pp} + (N_m - 1)(\lceil \log_2 m_i \rceil t_{pp} + t_{pp}) + t_d$$
$$= (\text{set encoders}) + (\text{propagate through encoders}) +$$
$$(\text{set subtractors}) + (\text{propagate through subtractors}) +$$
$$(\text{propagate through map}) + (\text{set adders and subtractors}) +$$
$$(\text{propagate through adders and subtractors}) + (\text{detect}).$$

(5) Influences Of Technology On Processor Performance

As mentioned previously, the expressions for maximum processor time derived for the various modular processors provide a means of evaluating the effects of certain technological approaches to certain types of computation problems.

As an example, suppose that the maps were implemented with wire interconnections and the permutation primatives were constructed using field effect transistors (FET's) as switching devices as shown in Fig. 14. As discussed previously, the other building blocks can be constructed from these basic elements. If the set time, t_s, is assummed to be 10 nanoseconds and the propagation time, t_p, of each building block is assummed to be 5 ns. then the throughput rate for inner products of vectors with 100 elements of arbitrary accuracy would be approximately

$$r = 1/[\ 3(\ 10\ ns\ ) + 100(\ 5\ ns\ )]$$

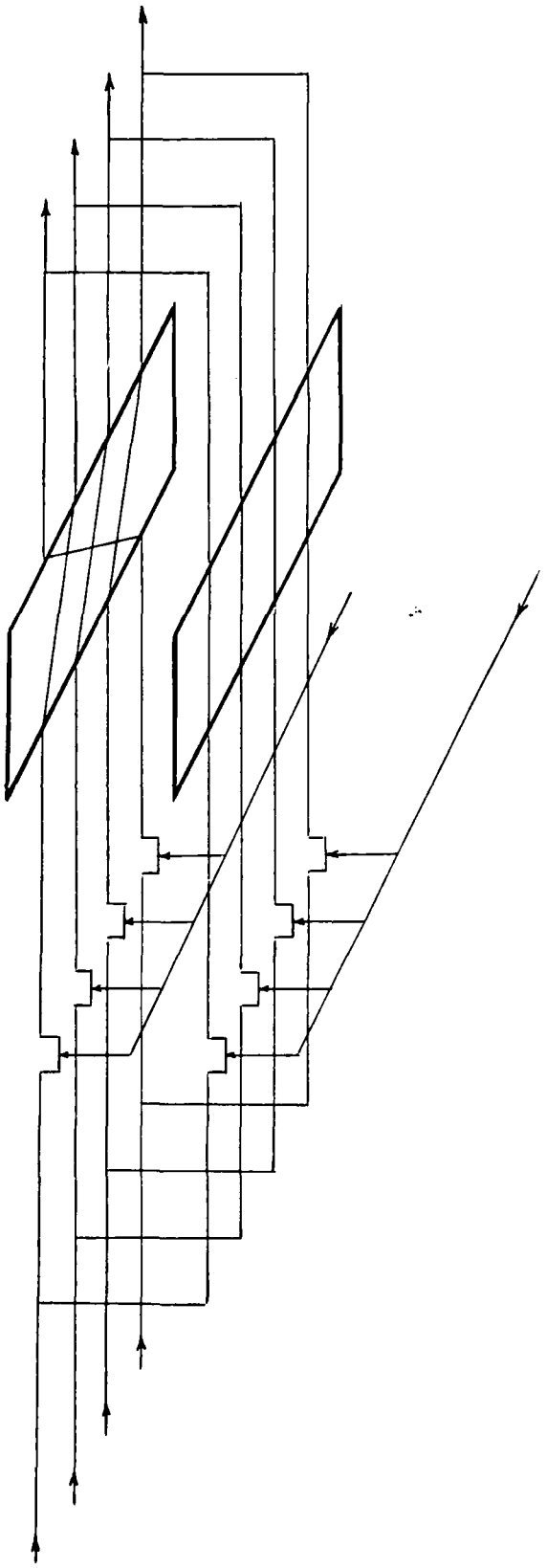


FIG. 14: FET PERMUTATION PRIMITIVE

$= 2$ mhz.

This would be equivalent to about 2 million inner products a second. Since each inner product consists of 100 multiplications and 102 additions this is equivalent to $202 \times 2 \times 10^6$ or about 400 million arithmetic operations a second.

The performance of an optical version of the same processor can be examined in a similar manner. If Bragg coupler technology is considered, then a set time for each building block of 30 ns and a propagation time of 200 ps can be assumed. This would result in a throughput of approximately

$$r = 1/[\ 3(\ 30 \text{ ns}\ ) + 100(\ 200 \text{ ps}\ )]$$

$$= 9 \text{ mhz.}$$

This rate represents 9 million inner products a second, which is is equivalent to $202 \times 9 \times 10^6 = 1{,}800$ million arithmetic operations a second.

The high throughput of residue processors used as examples is not due to the technology but rather the architecture. The set times of 10 and 30 ns are quite slow compared to the switching times of present electronic logic. The throughput is due to both a mathematical and structural advantage. Mathematically, not having to deal with carries greatly speeds up addition and multiplication. Structurally, the processor is highly parallel and very effectively pipelined.

D. Modular Processors Pipelined By Banks (The Turtle)

Modular Processors can also be pipelined by banks. This approach is referred to as the "turtle" or as a parallel number theoretic processor. Its structure favors different technological traits. The rabbit approach relies on speed while the turtle approach relies on parallelism to achieve large throughputs.

The overall structure of such processors is shown in Fig. 15. The data to be processed is fed to the modular encoders. The encoded values are then given to modular processors. The modular results are then fed to a residue-to-mixed-radix converter. The mixed radix equivalent is then given to a mixed-to-normal radix converter that produces a normal radix equivalent of the answer to the desired computation.

The processor relies entirely on table lookup. The tables can be implemented with read only memories. A modular 5 addition table is shown in Table 8. Neither the operands nor the sum ever exceed 4 in value. The operands and sum can thus each be expressed with 3 bits. This table can be implemented with a read only memory by combining the 3 bits of each operand to form an address and storing at that location a 3-bit representation of the sum. Such a table is shown in Table 9. As an example, to add 4 and 3 these operands are first expressed in binary as 100 and 011. These are combined to form the address 100011. Stored at this location in the memory shown in Table 9 is 010, which is the modulus 5 equivalent of the sum of 4 and 3.

Similar tables can be constructed to perform modular subtraction, multiplication , and polynominal transforms for any modulus.

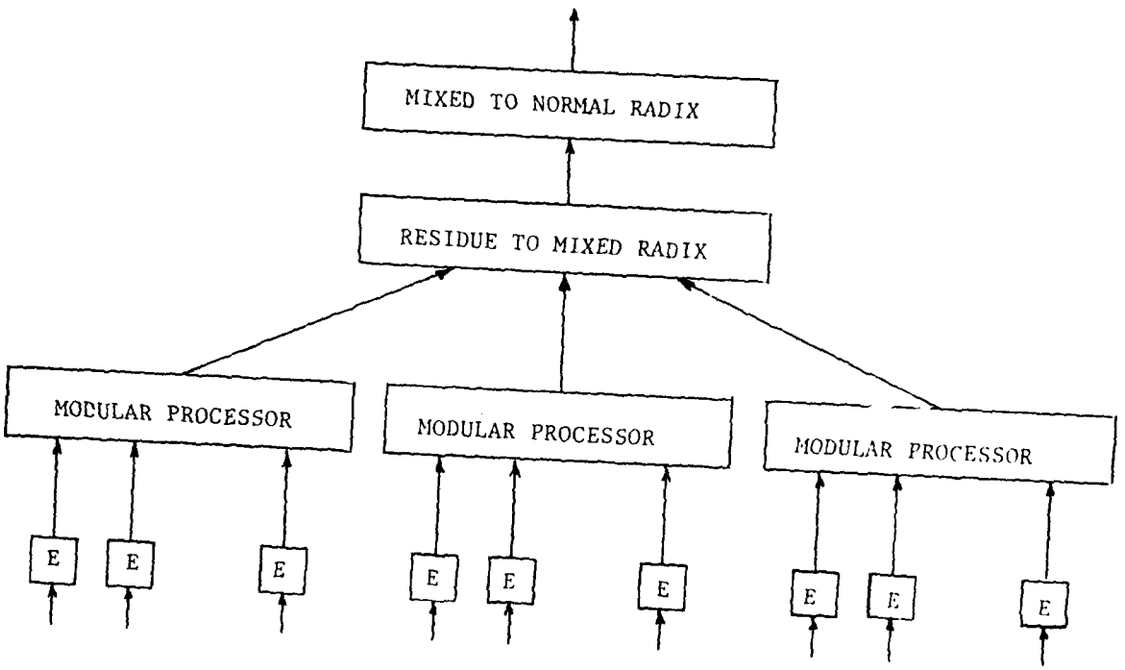


FIG. 15: TURTLE PROCESSOR

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

TABLE 8: MODULUS 5
ADDITION TABLE

ADDRESS	CONT
000000	X000
000001	X001
000010	X010
000011	X011
000100	X100
000101	XXXX
000110	XXXX
000111	XXXX
001000	X001
001001	X010
001010	X011
001011	X100
001100	X000
001101	XXXX
001110	XXXX
001111	XXXX
010000	X010
010001	X011
010010	X100
010011	X000
010100	X001
010101	XXXX
010110	XXXX
010111	XXXX
011000	X011
011001	X100
011010	X000
011011	X001
011100	X010
011101	XXXX
011110	XXXX
011111	XXXX
100000	X100
100001	X000
100010	X001
100011	X010
100100	X011
100101	XXXX
100110	XXXX
100111	XXXX
101000	XXXX
101001	XXXX
101010	XXXX
101011	XXXX
101100	XXXX
101101	XXXX
101110	XXXX
101111	XXXX
110000	XXXX
110001	XXXX
110010	XXXX
110011	XXXX
110100	XXXX
110101	XXXX
110110	XXXX
110111	XXXX
111000	XXXX
111001	XXXX
111010	XXXX
111011	XXXX
111100	XXXX
111101	XXXX
111110	XXXX
111111	XXXX

TABLE 9: ROM VERSION OF MODULUS 5
ADDITION

(1) Binary to Residue Encoders

Tables to perform encoding from binary to residue notation can be performed by tables constructed using three basic strategies.

One approach is by direct table lookup as shown in Fig. 16(a). The number to be encoded is used as the address while the content at this location is the desired modular equivalent. This approach is only practical for encoding relatively small numbers (less than 10 bits).

A second method involves a cascaded approach as shown in Fig. 16(b). The bottom most node performs a direct table lookup to encode a portion of the number while the subsequent nodes encode other portions and adds the equivalent of this new portion in a modular manner to the result from the previous node. This chaining process can be extended to incorportate any number of bits.

A final method involves a parallel approach as shown in Fig. 16(c). Portions of a number are encoded by direct table lookup. The equivalents of these portions are then added together in a modular manner. This method can be extended to incorporate any number of bits.

These three basic approaches to encoding can be modified to include the encoding of negative numbers represented in either sign magnitude or two's complement format.

Once the data has been encoded into their modular equivalents it can be used to perform computations in a modular manner.

(2) Summation Processor

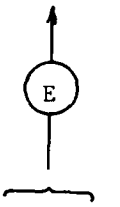


FIG. 16a:
DIRECT ENCODING

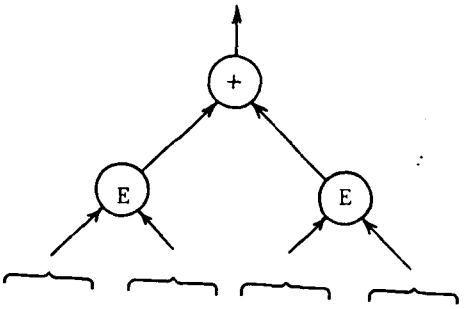


FIG. 16c:
PARALLEL ENCODING

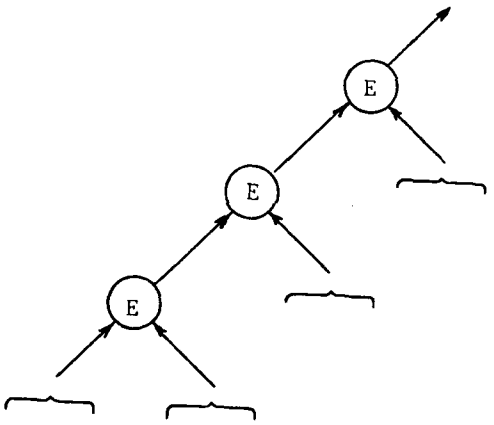


FIG. 16b
CASCADED ENCODING

The structure of a modular processor that performs summation for a particular modulus is shown in Fig. 17. Each of the nodes represents a read-only memory. The data entering the bottom of a node is used to form an address. The content of the node at that particular address emerges at the top of the node. If the modular equivalents of the 16 numbers to be summed are fed to the bottom row of nodes then the modular equivalent of their sum will eventually emerge from the root node (top most node). Such a processor can be constructed for any modulus by programming the ROMs in the proper manner.

(3) Inner Product Processor

A modular processor that performs inner products is shown in Fig. 18. The boxes represent encoders, the nodes with an X represent modular multiplication ROMs, and the node with a D represents a delay node *. The modular equivalents of the vectors to be processed are fed to the bottom row of nodes. The modular eqivalent of the inner product will emerge from the root node.

(4) Determinant Processor

A modular processor that evaluates determinants is shown in Fig. 19. The main difference from the inner product processor is that more multiplier nodes are needed. The modular equivalents of the matrix coefficients are fed to the bottom row of nodes. The modular equivalent of the determinate will emerge from the root node.

* A delay node duplicates the bits of its address on its output at a later time. This delay maintains synchronization between the various portions of a calculation so that they merge correctly.

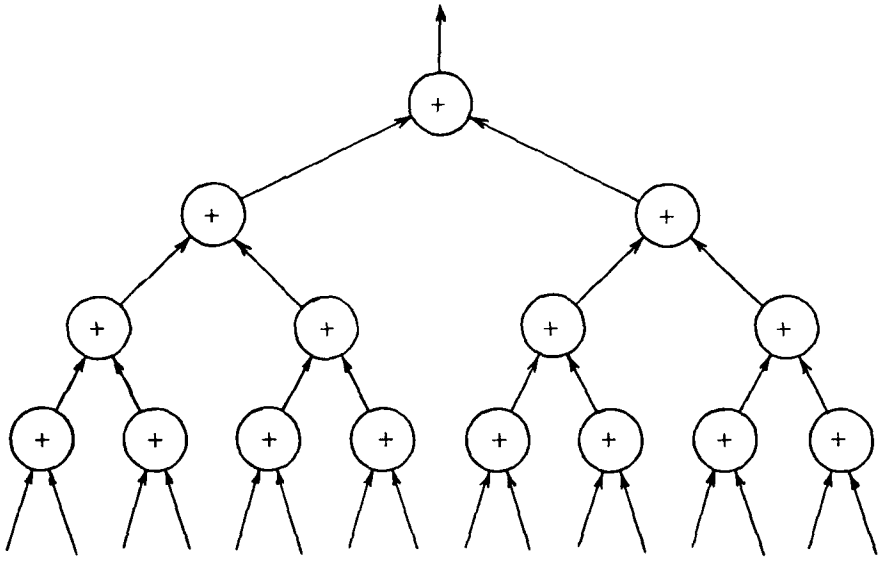


FIG. 17: MODULAR SUMMATION PROCESSOR

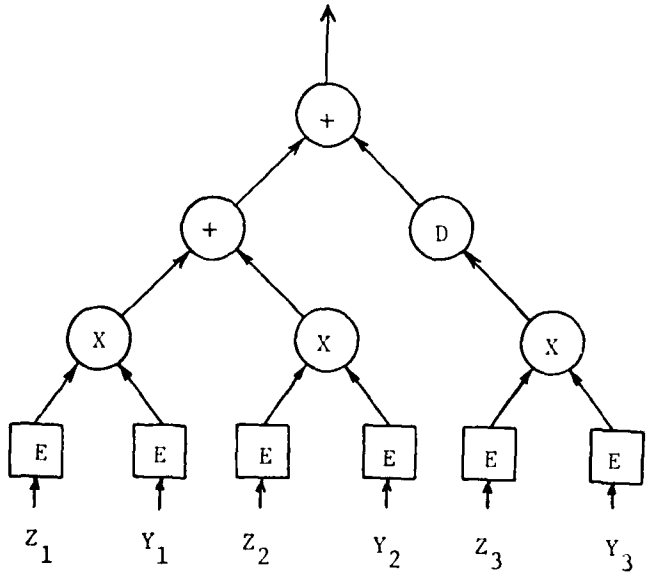


FIG. 18: MODULAR INNER PRODUCT PROCESSOR

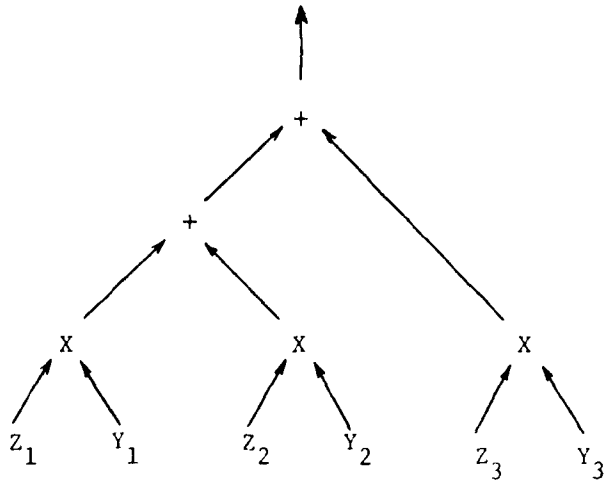


FIG. 18.1: EXPRESSION EVALUATION TREE
OF AN INNER PRODUCT

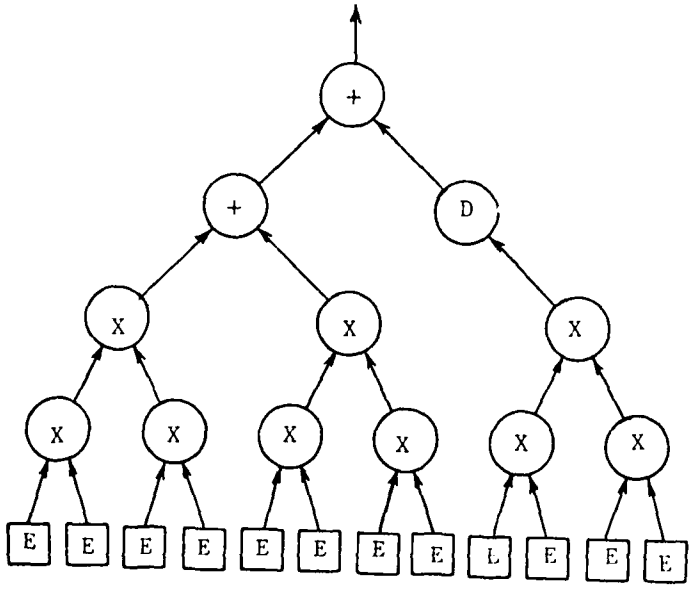


FIG. 19: MODULAR DETERMINANT PROCESSOR

(5) Squared Distance Processor

A modular Processor that computes the sum of the squares of the differences between the components of two vectors is shown in Fig. 19.5. The nodes denoted with a P perform the integer polynominal transform $P(X) = (X-Y)^2$ *. The modular equivalents of the vectors are fed to the bottom row of nodes. The modular equivalent of the sum of the squares of the difference of the vectors will emerge from the root node.

(6) Other Computations

To generalize, a modular processor can be designed to perform any combination of additions, subtractions, multiplications, or integer polynominal transforms. The structure of such a processor follows directly from the expression evaluation tree of the desired computation. The expression evaluation tree of the inner product processor in Fig. 18 is shown in Fig. 18.1. The only difference is that a delay node has been inserted to equalize the terminal path lengths to insure proper synchronization of the different portions of the computation.

(7) Conversion From Residues To Mixed Radix

A given computation is done in a modular manner by several different modular processors. These modular results are then rewoven together to construct a mixed radix version of the answer. This mixed radix conversion can be performed by various integer polynomial transforms [Ref. 1, p 17-25]. The structure of such a converter is

* Other processors can be designed using more complex integer polynominal transforms. The tables required are no more complex that those required for addition.

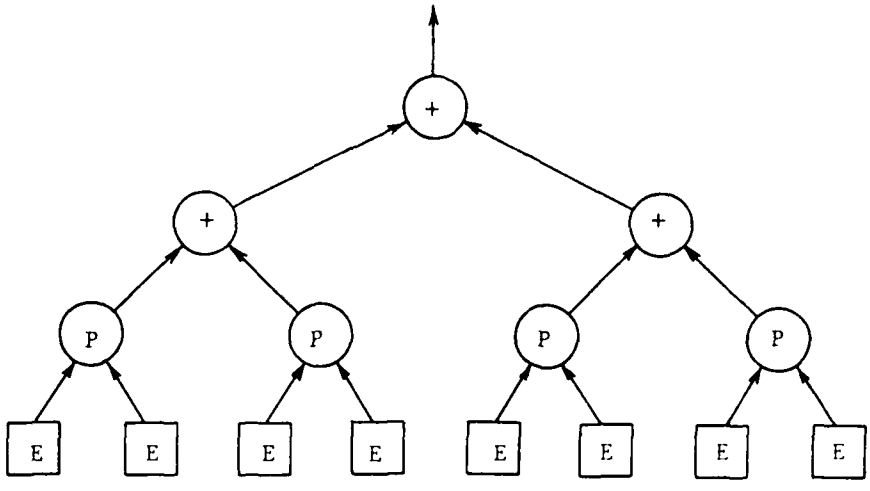


FIG. 19.5: MODULAR VECTOR DISTANCE PROCESSOR

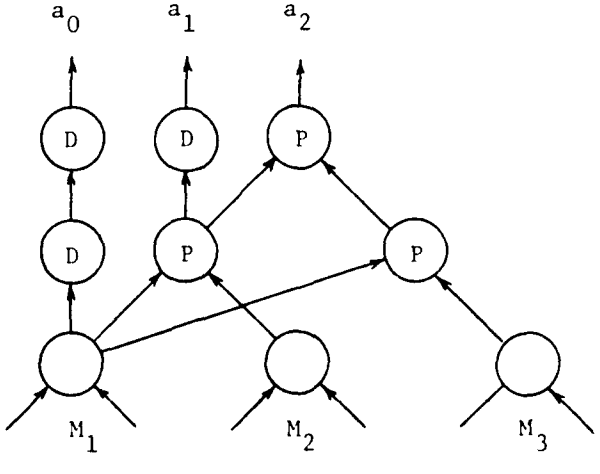


FIG. 20: RESIDUE TO MIXED RADIX CONVERTER

shown in Fig. 20. The nodes on the bottom row represent the root nodes of three modular processors. The nodes denoted with a P perform the required polynominal transforms. Delay nodes temporarily store the coefficients as they are produced. What emerges are the mixed radix coefficients of the answer to the desired computation.

(8) Conversion From A Mixed To A Normal Radix

The mixed radix version of the answer is sufficient for sign and relative magnitude determination. In some situations it is desirable to further convert the mixed radix into a normal radix number. An overview of such a converter is shown in Fig. 21. Each value of each mixed radix coefficient has a normal radix equivalent. These equivalents are recalled from storage and added to produce a normal radix equivalent. To simplify this addition, a carry save adder strategy is employed. This reduces, without carries, the equivalents to be added to only two. A carry propagate adder is then used to add this final pair. The storage, carry save adders, and carry propagate adder can be implemented in a bit slice format with read-only memories.

The structure of the storage section is shown in Fig. 22. The nodes on the bottom row are the output nodes of the residue to mixed radix converter shown in Fig. 20. The mixed radix coefficients are distributed to various storage nodes, denoted with an S, that store multibit slices of the normal radix equivalents. The top left three nodes store bits 11 through 8, 7 through 4, and 3 through 0 of the equivalent of the mixed radix coefficient a_0. The other nodes store multibit slices

MIXED TO NORMAL RADIX

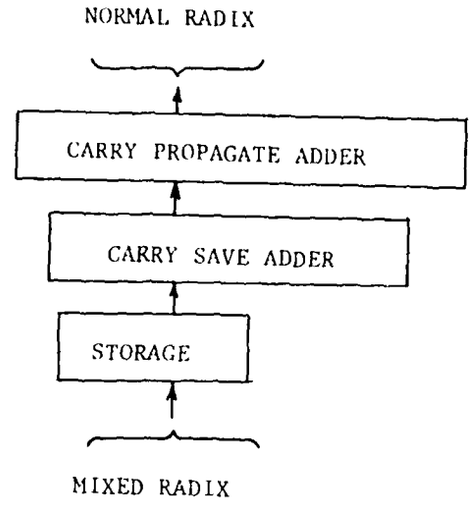


FIG. 21: OVERVIEW OF MIXED TO NORMAL RADIX CONVERTER

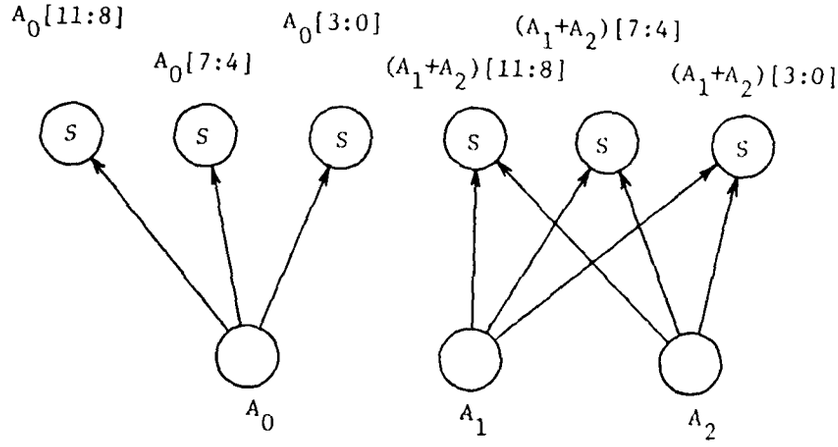


FIG. 22: STORAGE OF NORMAL RADIX EQUIVALENTS

of the sum of the equivalents associated with the mixed radix coefficients a_1 and a_2. This pairing and pre-addition technique reduces both the storage that is needed and the number of equivalents to be added.

The carry save adder section, which reduces a 3 number sum into a 2 number sum in parallel without using carries, is shown in Fig. 23. The nodes denoted with an A perform conventional binary addition on multibit operands. These adders are implemented with tables. A table for 2 bit operands is shown in Table T15 of the appendix. Each adder node handles a 2 bit slice from each of the three numbers to be added. It produces a 2 bit sum and a 2 bit carry. The sum bits from all the adders are combined to form one number while all the carry bits form another number. Two zero bits are padded onto the number synethized from the carry bits to perserve the proper significance of these carry bits. A sum of 3 numbers can thus be reduced to a 2 number sum. Several such carry save adders can be used to reduce a sum of many numbers to only a sum of two numbers.

The remaining 2 numbers are added with a carry propagate adder. A carry propagate adder is shown in Fig. 24. The two numbers are added in multibit slices from least to most significant slice. This allows a carry to propagate between the slices. The nodes denoted with an A are the same binary adders used in the carry save adder. The nodes denoted with a D are delay nodes to delay slices of the operands until they are needed. Delay nodes are also used to delay slices of the sum so that they emerge in synchronization.

(9) Synchronous And Asynchronous Turtle Processors

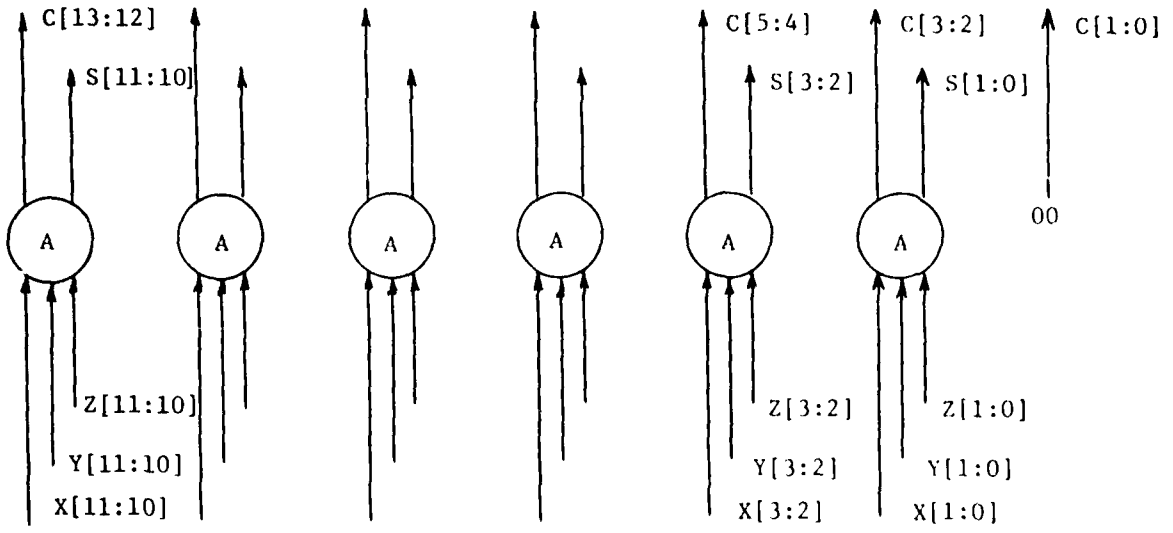


FIG. 23: CARRY SAVE ADDER

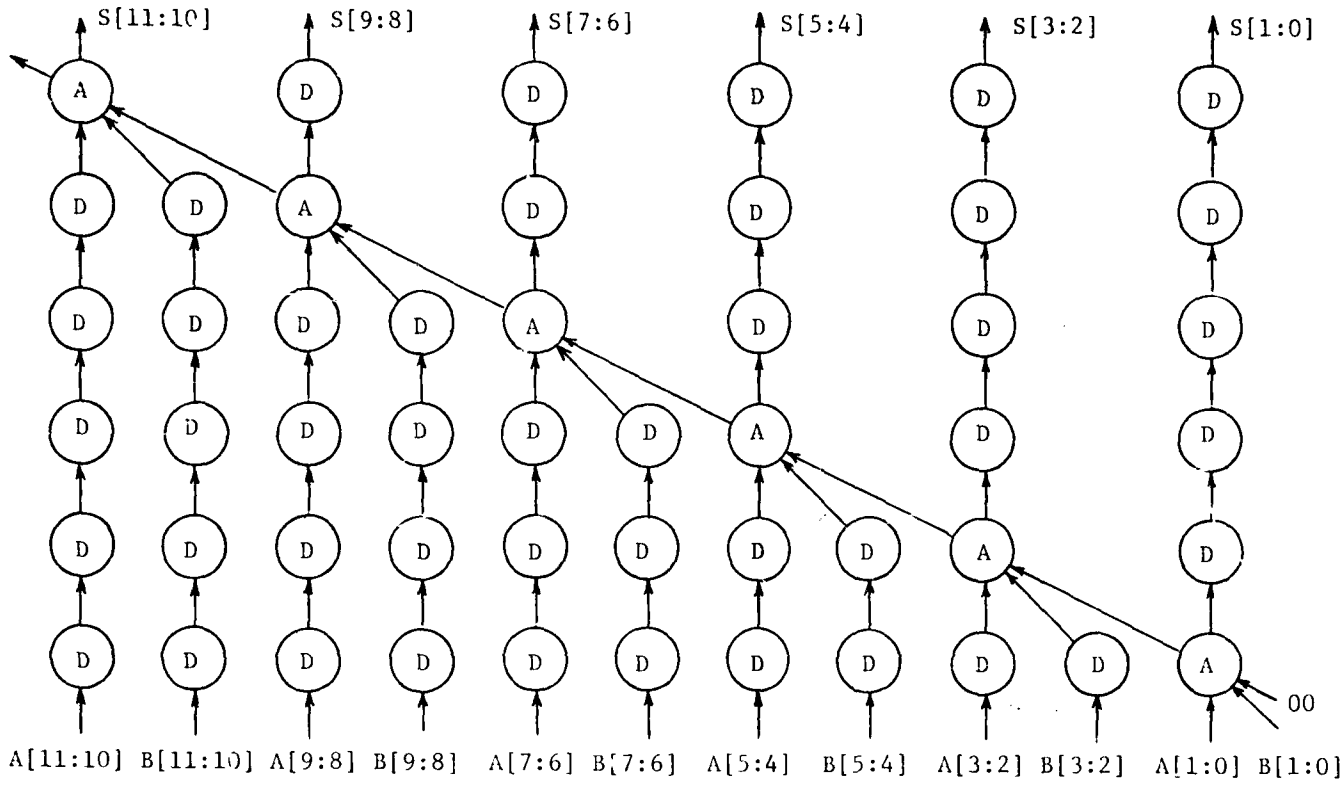


FIG. 24: CARRY PROPAGATE ADDER

An overview of a turtle processor designed to perform summation is shown in Fig. 25. The bottom row of nodes consists of encoders. The next two rows consists of adders. Each of three clusters of nodes at the bottom represent different modular processors. The fourth and fifth row form the residue-to-mixed radix converter. The sixth row contain the storage units for the mixed-to-normal radix converter. Since in this simple example there are only two normal radix equivalents to be added, a carry save adder section is not necessary. The top 6 rows of nodes form a carry propagate adder.

This version of the processor is designed for synchronious operation. The ROMs used in the processor must either be input or output latchable. The data to be processed is fed to the bottom row of encoders. On a clock signal these memories are read and the results are used as the inputs for the next row or ROMs. The data for another computation is then fed to the encoders. On each subsequent clock cycle the data of a particular calculation proceeds to subsequent rows of the processor in a Roman-phalanx- like manner. The results of each calculation will eventually emerge from this pipelined processor. The throughput rate would be the recriprocal of the ROM cycle time.

A version of the processor can also be designed for asynchronous operation. Such a structure is shown in Fig. 26. In this case the ROMs are not latachable. The data for a particular computation is placed at the encoders. The results just propagate through the system. There are many races but since there is no feedback or memory in the processor none of the races are critical. The answer is derived in a Darwinian manner. The unit can be viewed as one large combinatoric circuit.

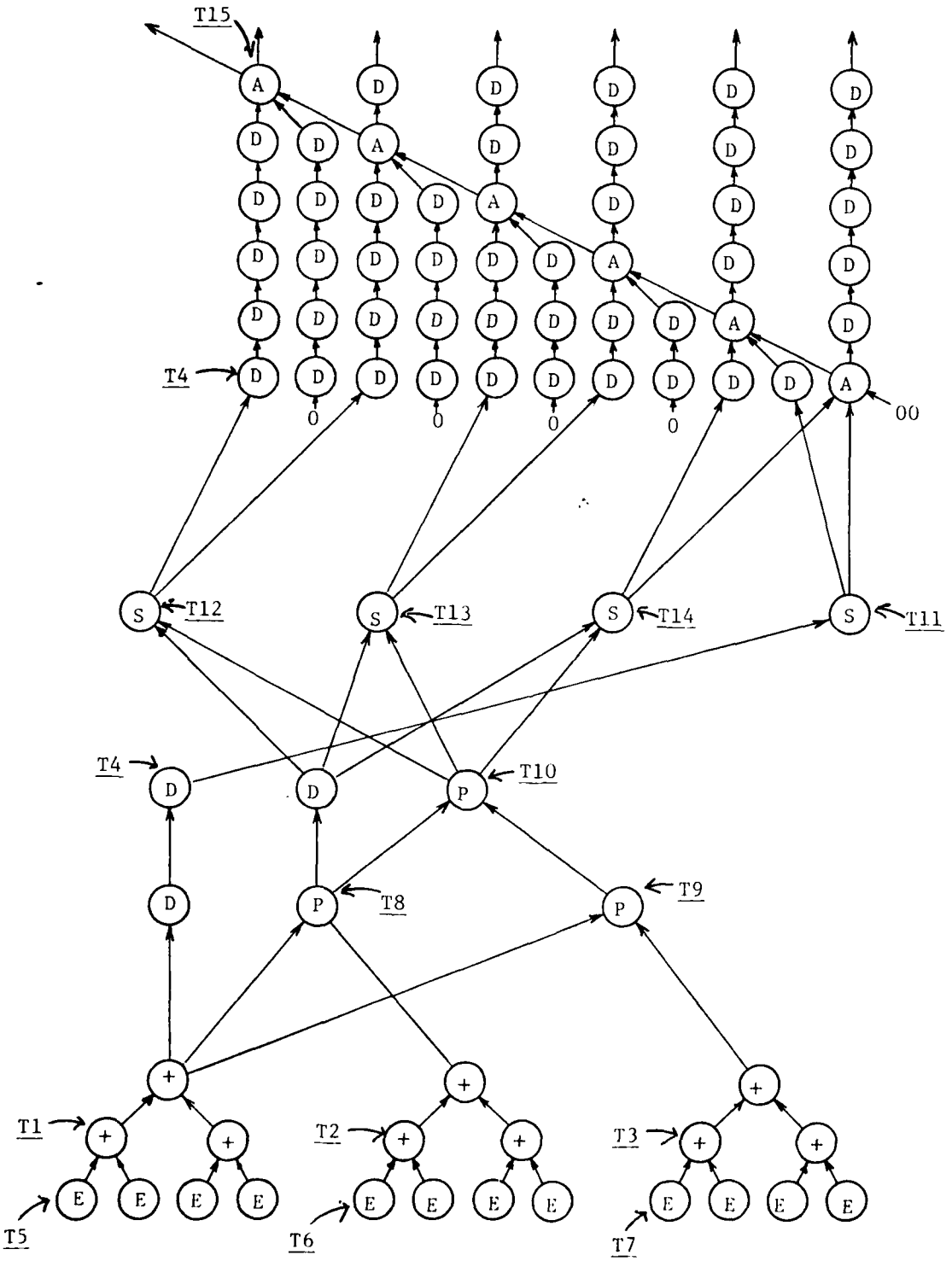


FIG. 25: SYNCHRONOUS NUMBER THEORETIC PROCESSOR

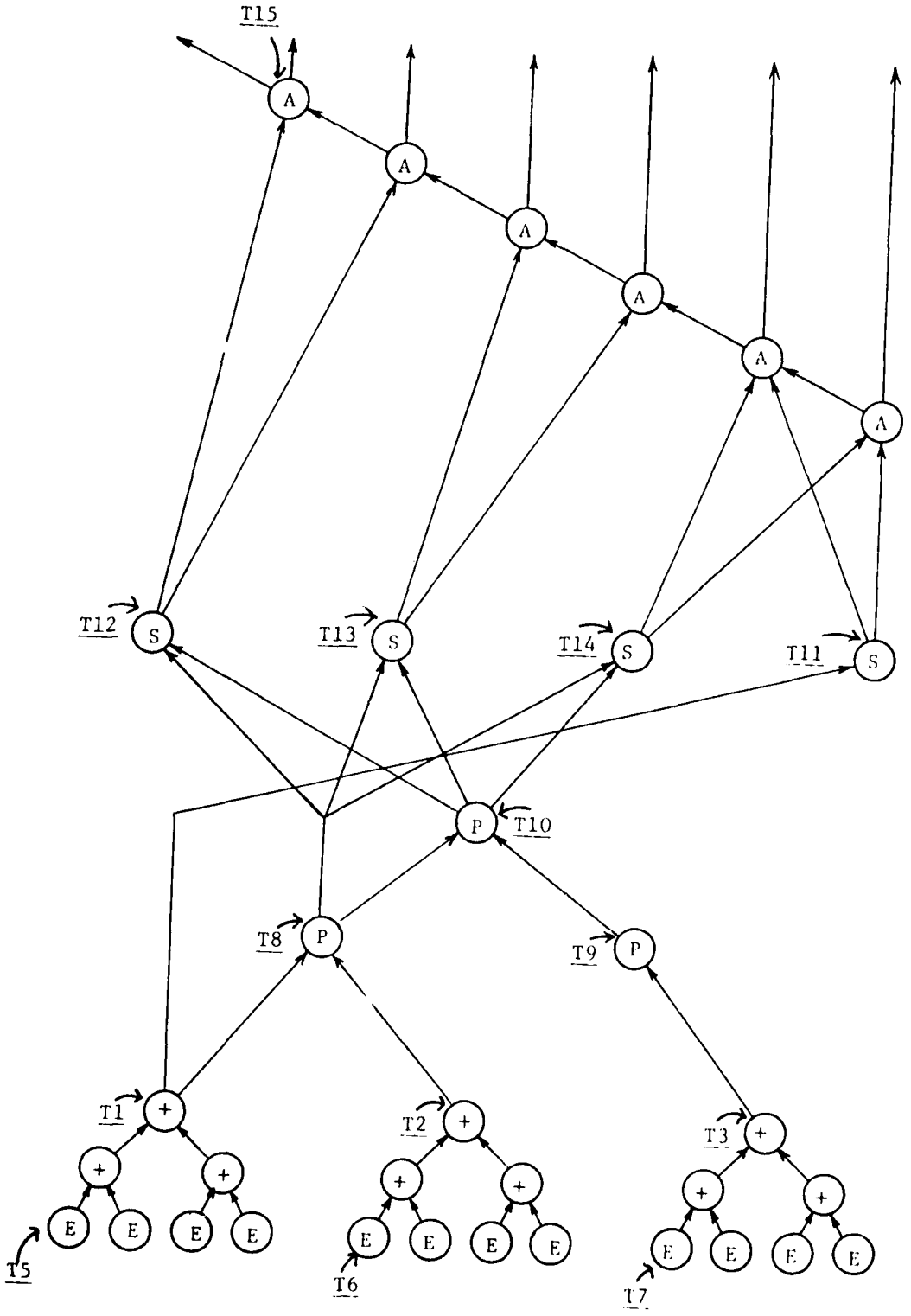


FIG. 26: ASYNCHRONOUS NUMBER THEORETIC PROCESSOR

Any computation involving any combination of additions, subtractions, multiplications, or integer polynominal transforms can thus be accomplished with a combinatoric circuit.

(10) Example Of A Turtle Processor

As an example of how such turtle processors would operate, the tables representing each of the nodes shown in Fig. 25 and 26 are included as tables in the appendix. An example of the sum of 13 + (-11) + 7 + (-17) is shown in Table 10. In six bit two's complement, this sum is expressed as 001101 + 110101 + 000111 + 101111. Table T5 is used to translates these values into their modulus 5 equivalents of 011, 100, 010, and 011. 011 and 100 are then combined to form the address of 011100 which is translated by table T1 into the modular sum of 010. 010 and 011 are summed in the same manner to produce 000. These two results are then summed with table T1 to produce 010. Tables T6 and T2 perform this same procedure for modulus 7. The result is 110. Tables T7 and T3 perform this same procedure for modulus 8. The result is 000.

The modulus 5, 7, and 8 results of 010, 110, and 000 are converted to mixed radix by tables T8, T9, and T10. The modulus 5 and 7 results are combined to form the address of 010110. Table T8 translates this into 101. The modulus 7 and 8 results are translated by table T9 into 110. These results are then combined to form an address of 101110, which is translated by table T10 into 111. This produces the mixed radix coefficient a_0 = 010, a_1 = 101, and a_2 = 111. Table T11 then produces the normal equivalent of a_0, which is 0010. Tables T12, T13, and T14 produce slices of the normal radix equivalent associated with the pair

EXAMPLE

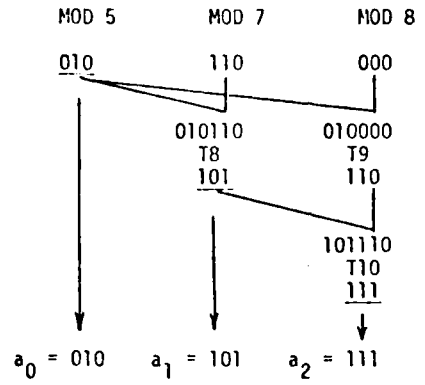
13 + (-11) + 7 + (-17) =

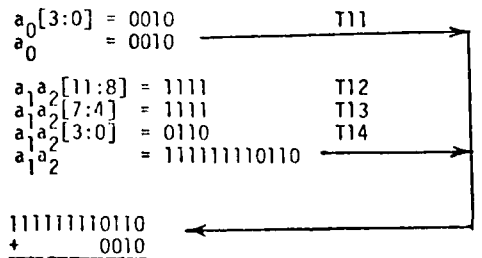
001101 + 110101 + 000111 + 101111

MOD 5 011 + 100 + 010 + 011 T5
010 + 000 T1
010 T1

MOD 7 110 + 011 + 000 + 100 T6
010 + 100 T2
110 T2

MOD 8 101 + 101 + 111 + 111 T7
010 + 110 T3
000 T3





00	00	00	00	01	00
11	11	11	11	01	10
+00	00	00	00	00	10
11	11	11	11	10	00

T15

111111111000 = -8
= 13 + (-11) + 7 + (-17)

TABLE 10: EXAMPLE OF NUMBER THEORETIC PROCESSOR

of coefficients a_1 and a_2. The result is 111111110110. These two normal-radix equivalents are then added with binary adder nodes, as represented by Table T15. The least significant slice of this sum is 00, 10, and 10. The 00 is the result of a null carry in for the least significant slice. These operands form an address of 001010. Table T15 translates this into 0100. The first two bits are used as a carry slice while the last two bits form the sum slice. The carry is used as an operand in the next more significant slice. The operands of this slice are 01, 01, and 00. Table T15 produces a result of 0010. 00 is used as a carry while 10 is used as the sum. This process is continued for the other slices. The result is 111111111000, which is -8 in 12 bit two's complement. This is the result of the desired sum of 13 + (-11) + 7 + (-17).

(11) Accuracy

The processor is constructed entirely from 64 by 4 bit ROMs. It has a range of -140 to 139. By using larger ROMs larger moduli can be represented and thus larger ranges can be achieved.

If 1024 by 6 bit ROMs are used, then the moduli 32, 29, 27, 25, 23, 19, 17, 13, 11, and 7 can be used giving a range of 144,403,552,893,600 which is about 2^{47}.

If 4096 by 6 bit ROMs are used, then the moduli 64, 61, 59, 57, 53, 51, 49, 47, 43, 41, 37, 31, 29, 25, 23, 19, 17, 13, and 11 can be used. This would give a range of 127,290,734,521,737,197,468,723,265,600 or about 2^{96}. Larger ROMs can be used to achieve even greater ranges if desired.

The concept of range is different in a residue number system. An intermediate computation may exceed this range. It is only necessary for the final answer to be within the range before it is converted into a mixed radix number. In the literature this property is called "compute through overflow". Thus the range need only be sufficient to represent the answer. This can greatly reduce the range required for certain types of computations.

(12) Throughput

A processor designed to compute inner products of 100 element vectors where each element is 20 bits long, can be constructed entirely out of 1024 by 6 bit ROMs. If the cycle time of each ROM is 300 nanoseconds then 3.3 million inner products can be performed a second (1/[300 ns.]). Since each inner product consists of 100 multiplication and 102 additions this is equivalent to 663.3 million arithmetic operations a second. The latency of each inner product would be 12.3 microseconds.

A processor using the same ROMs can be designed to perform inner products on 1000 element vectors, where each element is 18 bits, at the same throughput rate of 3.3 million inner products a second. In this case each inner product consists of 1000 multiplications and 1013 additions. This would be a throughput of 6.71×10^9 arithmetic operations a second. The latency would be 13.5 microseconds #.

(13) System Fabrication

As a rough reference of throughput an IBM 370 can process about 4 million instructions per second. Processors such as the CRAY - 1 and the ILLIAC IV can do up to 100 million instructions per second in short bursts.

These processors can be constructed in a modular manner. A large portion of the design of a conventional processor is consummed with the layout. The typical procedure involves partitioning the circuit, laying out the boards and specifying the blackplane interconnections. The turtle processor is constructed entirely with 2 input and 1 output nodes, 3 input and 2 output nodes, and their interconnections. The nodes and interconnection cables can be mass manufactured. The basic modules are shown in Fig. 27. The top two modules represent different types of nodes. On the bottom row the 1 input, 2 output module is a forked cable which duplicates a signal. The 1 input, 1 output module is an amplifier. Standardized cables would carry the information signals as well as utilities, such as the clock signal and the power between the modules. The construction of such processors is reduced to specifying the pattern of the nodal interconnections and the programming of the ROM of each node.

E. What Has All This To Do With Optics?

The turtle processor performs many table lookups in parallel during each cycle. Optical methods capable of performing many table lookups in parallel can be conceived. The fundamental device in such an optical processor would be an optical ROM. It is important for such a device to produce an output that can be used as an input of another such ROM. In other words, it should "produce what it eats". In optics the basic information-carrying mechanisms are intensity , phase, frequency, or spatial distributions of these quantities. A ROM based on intensity would have to have as inputs various intensities and produce a predetermined output intensity. No convienent implementation based on either intensity, phase, or frequency has yet been found.

(1) Optical ROMs

The most promising approach for an optical ROM is based on embedding information on the spatial domain. The input operands of such a device would be intensity distributions of light, $I(x,y)$, and the output of the ROM would also be an intensity distribution of light that could be used as an input operand for another such ROM.

A block diagram of an optical ROM is shown in Fig. 28. Two or more images, $I_{in}(x,y)$, would be ANDed together to form an unique image, $I_{addr(x,y)}$. This image would be used to address an associative memory to produce an output image, $I_{out}(x,y)$. This image could then be used as the input for another such ROM.

(2) Optical ROM Addressing

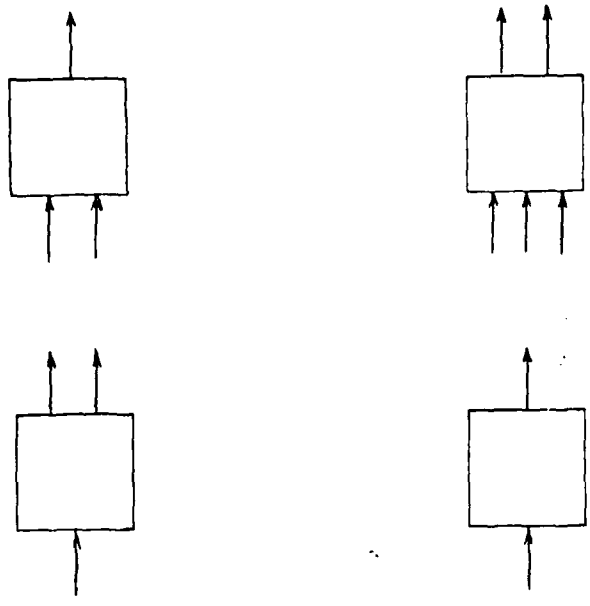


FIG. 27: MODULES FOR NUMBER THEORETIC PROCESSORS

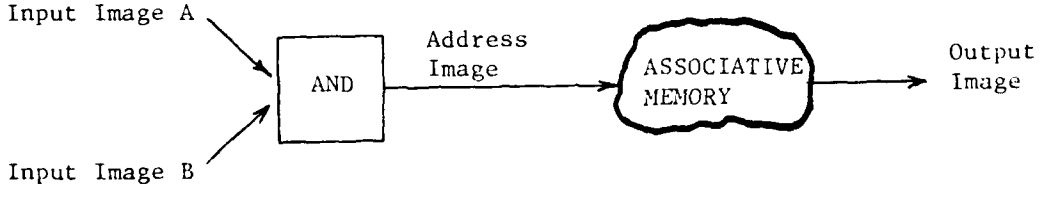


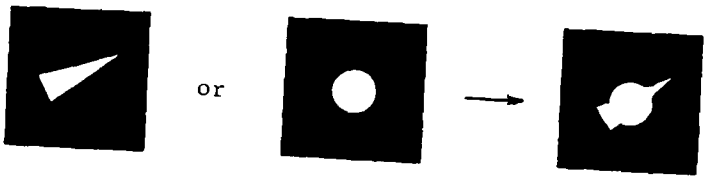
FIG. 28: BLOCK DIAGRAM OF AN OPTICAL ROM

The fundamental operation of locating or addressing anything anywhere, whether it is in a computer memory or on a road map, is the "AND" operation. An address of 011 in a computer means that the 2^2 bit of the address is 0 and the 2^1 bit is a 1 and the 2^0 bit is a 1. Unfortunately, optical ANDs cannot be performed in a convienent manner; however an optical "OR" can.

If two or more binary images, I(x,y), are projected on a common surface and thresholded, then the result will be the union or "OR" of the two images, as shown in Fig. 29. With the help of some negations (contrast reversals), the ORs can be used to perform ANDs by means of DeMorgan's law , A AND B = NOT(NOT A OR NOT B), as shown in Fig. 29. The images to be ANDed are first inverted in contrast. These images are then ORed by projecting them on a common surface and thresholding. The image on the surface is then inverted in contrast. This results in the AND of the orginal images.

It is important that the AND of the input images form a unique output image. Otherwise [illegible] would confuse the memory. This uniqueness can be [illegible] the input images as shown in Fig. 30(a). Suppose that [illegible] horizontal dark bars could each only be in one of five [illegible] the inversion of the OR of any of the possible vertical patterns with any of the possible horizontal patterns will result in a unique pattern. This pattern happens to be a point source that will provide a convienent address image for an associative memory. The five vertical and five horizontal patterns are sufficient to each represent modulus 5 operands.

Images of the form of a dark vertical or horizontal bars are



OPTICAL "OR"

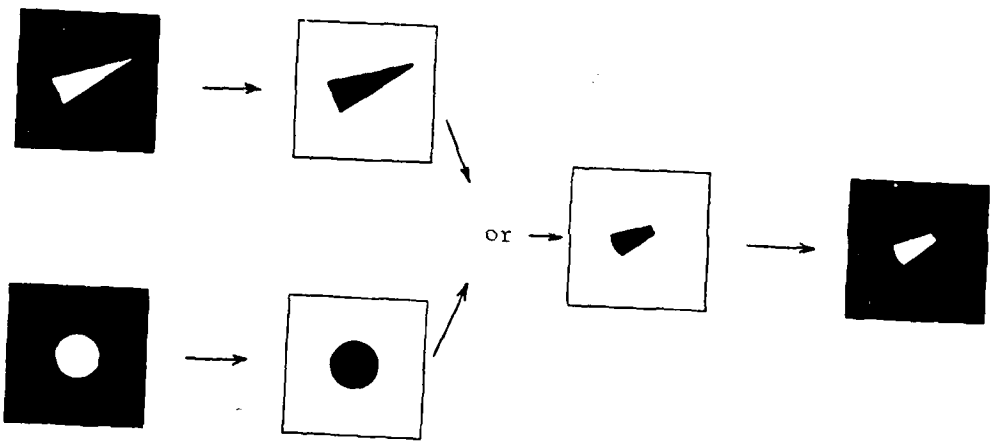


FIG. 29: OPTICAL "AND"

FIG. 30(a): UNIQUE INPUT IMAGES

FIG. 30(b): UNIQUE CYCLIC IMAGES

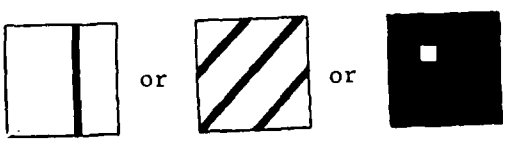


FIG. 30(c): OUTPUT IMAGES

suitible as inputs for any two input ROMs. This type of ROM is sufficient to construct encoders, modular processors, and the residue-to-mixed radix converter. If it is desired to completely convert the result back to a normal radix, then 3 input and 2 output ROMs would be required. This necessitates a larger variety of input and output images. Such images are shown in Fig. 30(b). They have a cyclic structure. If the frequencies of the images are pairwise relatively prime, then the OR, and negation and thresholding of such images will also form a unique point source image suitable for use as an address.

To generate the address image, the inversion of the OR of the input images has to be accomplished. Several physical mechanisms can be used to accomplished this. One method uses the Hughes Liquid Crystal Light Valve.

One surface of the light valve is light sensitive. A distribution of light on this surface will modulate the electric field across the liquid crystal. This electric field in turn modulates the birefringence of the liquid crystal. This change in birefringence rotates the axis of polarization of a polarized read light beam impinging on the other surface. The reflected light is analyzed with a polarizer. Depending on the orientation of this analyzer the resulting image will either have normal contrast or reversed contrast when compared to the image on the input surface. The light valve would be used in a binary mode to achieve thresholding. It is assummed that the bright portions of each input image would be sufficient to saturate the photoconductor.

(3) Optical ROM Storage

The associative memory of an optical ROM must associate an address image with an output image. Such a memory can be holographic. The complexity of the memory is considerably simplified by using address images that are unique point sources. The output images to be associated with a point source address image are of the form shown in Fig. 30(c). If only mixed radix results are desired, then images of vertical or horizontal dark bars would be sufficient. If conversion to a normal radix is desired, then images with a cyclic or point-source pattern would be necessary.

The operation of such a holographic ROM is shown in Fig. 31. Two or more input images would be projected on to the surface of a Hughes Liquid Crystral Light Valve. This image would then be inverted in contrast to form a point source address image. This point source would address the hologram to produce an output image.

(4) System Integration

Many of these ROMs can be placed side by side on a common surface. Each portion of the surface denoting a ROM would OR its own input images. All the ORs of all the ROMs would be inverted at the same time. Each ROM will then have an appropriate address image for its corresponding hologram. Rather than having a mosaic of holograms, the holograms for all the ROMs can be amalgamated into one large hologram. This can be done since the address images of each ROM remains spatially unique even though the ROMs are placed side by side. Each point source address image will produce an output image. Spatial offsets can be incorporated into these output images. The output images can be positioned to be the

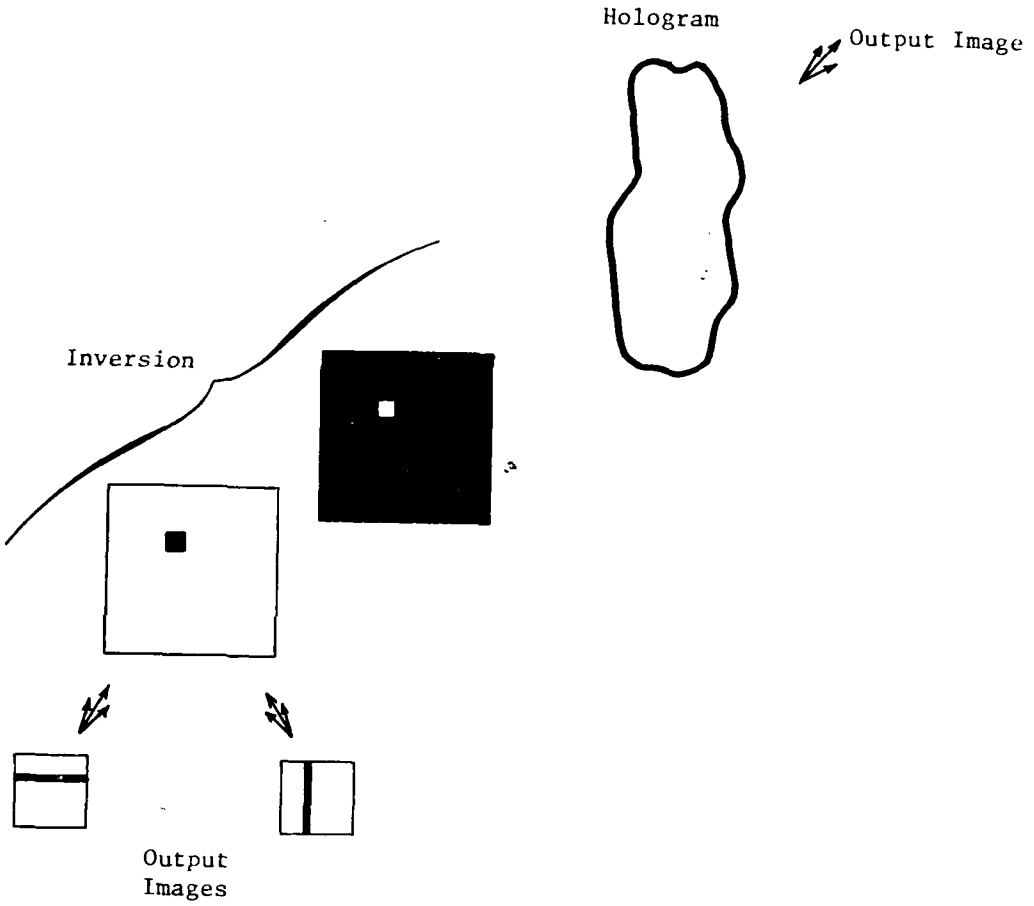


FIG. 31: HOLOGRAPHIC ROM

input images of other optical ROMs. The holograms would thus not only store the information but also distribute it.

In order to construct an optical turtle processor, the output of the ROMs must be used as inputs of other ROMs. What is desired conceptually is shown in Fig. 32. The output of the ROMs would be delayed a sufficient amount of time to allow the inversion mechanism to recycle. These delayed output images would then be used as input images. Unfortunately the slow recycle time of the inversion mechanism makes this approach impractial.

An approach using a twin set of ROMs is shown in Fig. 33. One group of ROMs is read and used to provide the input to the other group, and vice versa. The light passing through the hologram on the top left produces input images for all the ROMs on the light-sensitive side of the light valve. A polarized read light reflected from the light valve and analyzed by another polarizer produces a contrast inverted image of the entire surface. These point sources, reflected by a mirror, are used to address the hologram on the bottom row. This produces the input images for all the ROMs on the bottom light valve. Another read light is used to generate the inverse of this image, which produces the address images for the hologram on the top row.

The two ROMs with their light valves are organized in a two cycle approach. One light valve is used to write the other, and then vice versa. The delay or latency of the light valve is used as temporary storage. This keeps the system going, much like how a flywheel is used in a two cycle engine. (This configuration is referred to as a "torus turtle".) A three cycle or Wankel approach using 3 ROM banks can be

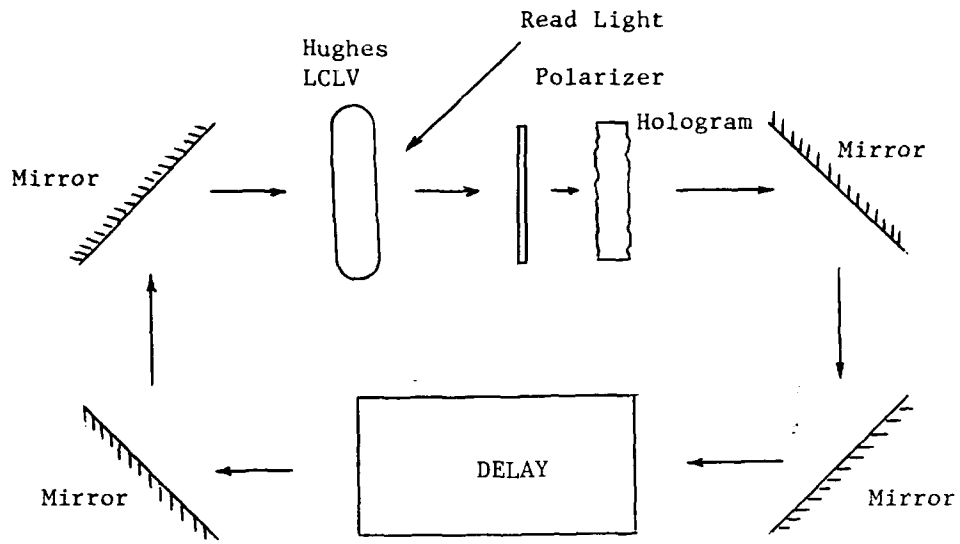


FIG. 32: OPTICAL TURTLE USING DELAY

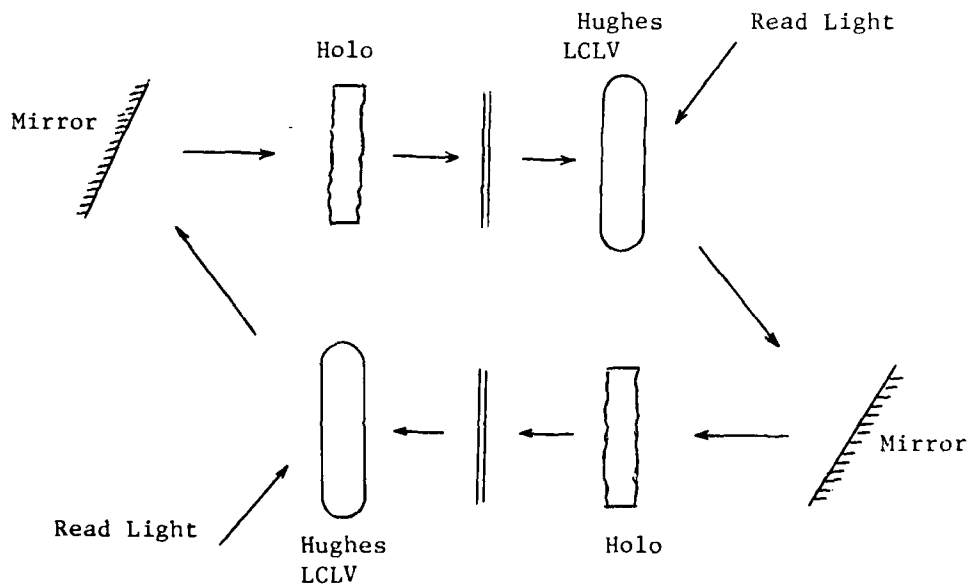


FIG. 33: OPTICAL TURTLE USING TWIN ROMS

considered if the latency is not sufficient. This approach can be extended to n cycles, as might be necessary to accommodate the long recovery time of the inverting mechanism in some technologies.

The optical processor as shown in Fig. 33 can be simplified by using reflective holograms to replace the mirrors as shown in Fig. 34. These reflective holograms can be constructed to be oriented at an arbitrary angle. This would lead to a further simplification as shown in Fig. 35. This sandwich structure would be simpler, more stable, and more compact.

In all the optical turtle processor configurations the input data is represented by an image that is projected on a portion of the ORing surface of a light valve. The output is represented by the image that is reflected from a portion of the ANDing surface of the light valve. LED's and optical masks can be used to translate conventional input data into images of the form shown in Fig. 30(a). The output images of the form shown in Fig. 30(c) can be translated by photodetectors into conventional electronic output.

The question remains as to how many ROMs are possible. This depends on the storage capacity of the hologram. The situation is similar to that of holographic optical memories. The required spatial bandwidth of each ROM is minimized by the use of point source addresses. For a ROM representing a modulus of 32 a field of 32 by 32 possible point source address would be needed. Each of the point sources must be asociated with an image of 32 by 32 elements.

(5) Why Optics?

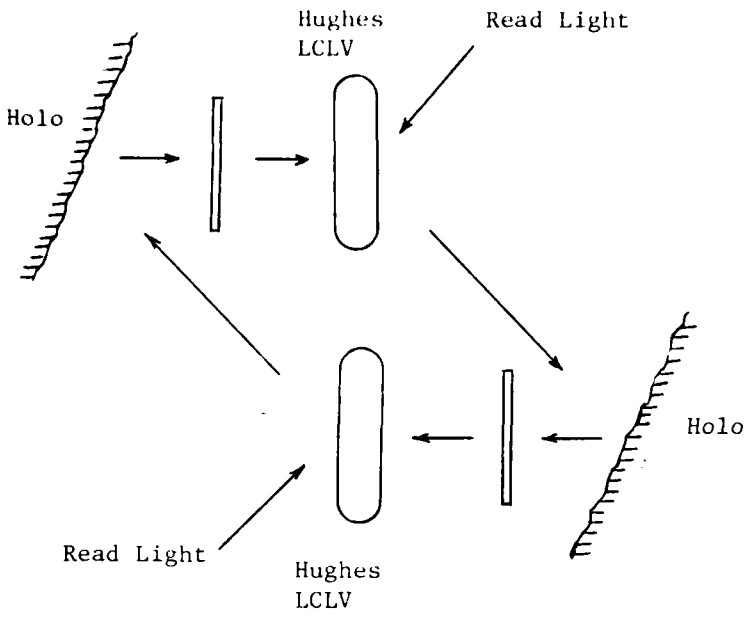


FIG. 34: OPTICAL TURTLE USING REFLECTIVE HOLOGRAMS

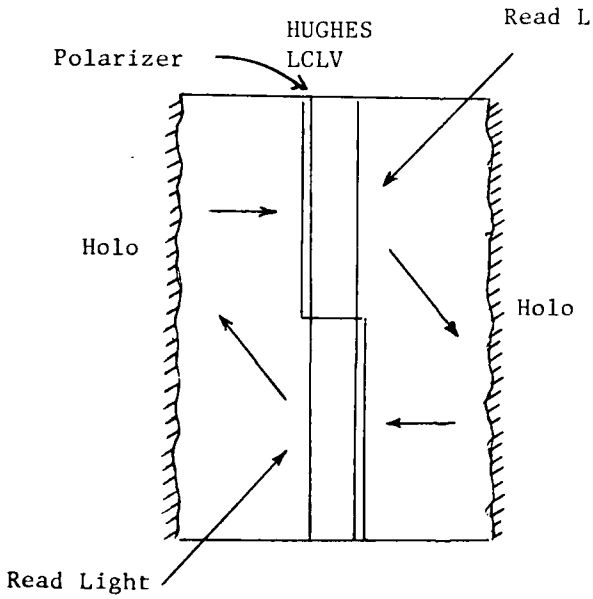


FIG. 35: SANDWICH VERSION OF OPTICAL TURTLE

The optical version has some unique advantages. The functions of addressing, storage, and distribution of information are all done in bulk. Making 100 optical ROMs should not be much more complex than making 10 ROMs. This approach could potientially benefit from economies of scale. Another advantage is that such a processor might be easier to mass manufacture, since reproduction of the holograms would be a photographic process rather than one of assembly, as electronic versions would be. A final point is that the use of optics considerably simplifies the communication problem. Optical signals can pass through each other without interference. They do not have to be shepherded around with wires as in the electronic version. Conventional wiring contributes an appreciable amount to the complexity and volume of current processors. A complete optical processor offers the hope of a more compact and easier-to-fabricate processor.

The optical version does have a serious disadvantage. The throughput rate of such a unit is basically the recriprocal of the cycle time of the contrast inverting mechanism. Currently the most available unit is the Hughes Liquid Crystral Light Valve (LCLV) which has a cycle time of about a millisecond. To make such a processor viable in terms of throughput, many points have to be processed in parallel to compensate for the slow cycle time.

The relative merits of an optical and electronic approach are depicted in Fig. 36. The increase of complexity of an electronic version with increased throughput can be predicted quite accurately. Admittedly, an optical approach has a high initial overhead, but its increase in complexity with increased throughput is unknown. The throughput at

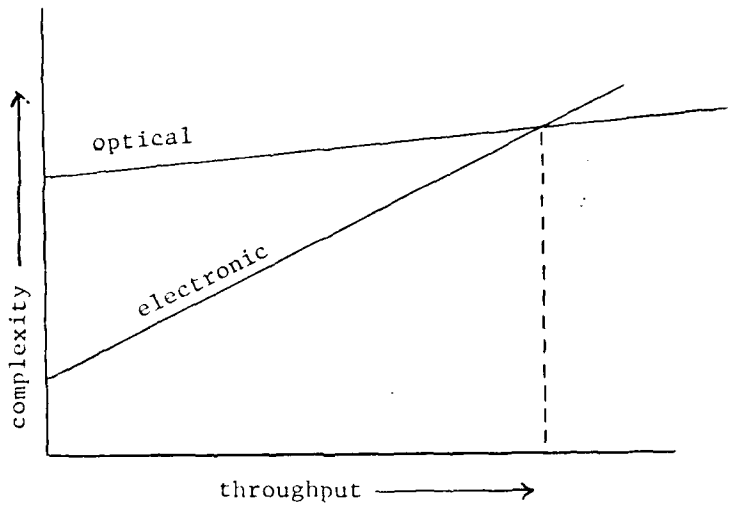


FIG. 36: COMPLEXITY VS. THROUGHPUT FOR ELECTRONIC AND OPTICAL IMPLEMENTATIONS

which optics overhead would be completely amortized is another open question. Electronic versions of the turtle processor will become more and more significant because of their outstanding throughput, cost effectiveness, and modularity. Larger and larger systems will be built. The possibility of an optical approach will continually haunt this growth.

III. RESEARCH ON RESIDUE ERROR DETECTION AND CORRECTION METHODS

A. Overview

The optics community is interested in research on the design of optical systems capable of data computation. One approach of present interest is based on the nature of arithmetic operations (addition and multiplication) in residue number systems [Ref. 1]. Residue number systems have desireable properties. Most significantly of these, perhaps, is that residue additions do not involve carry operations. The hope is that the natural parallelism of optical systems can exploit the fact that no carries are necessary.

Unfortunately, residue number systems also have very undesireable properties. The worst of these is: a small error in an analog process used to perform a residue addition may result in large error in the result of the residue addition. The intrinsic nature of this error process is discussed in the (3) On Noisy Residue Operations (Section B).

To date most research has focused on the design of physical systems that perform basic "residue operation" such as the mod operation or addition mod some moduli. There has been little published material on the development of a general mathematical system based on residue number systems capable of incorporating error detecting/correcting operations. The purpose of this paper is to make the following three contributions towards the development of such a general mathematical system.

First, a general mathematical system capable of characterizing (i)

any mathematical operation performed in a residue number system and (ii) all error detection/correction methods is constructed. This construction is undertaken in Sections B, C, and D and completed in Section E. The main result is found in the subsection (3) The Error Checking System (Section D).

Secondly, and importantly, a methodology for research into the intrinsic capabilities of all possible error detecting/correcting methods is proposed. This is done for the most part in Section E.

Finally, in Sections D, F, G are found examples of error detecting/correcting methods. The special example indicated in the abstract is in "Example: Special case of redundant encoding" (Section F).

In addition to the three contributions above, this chapter discusses in a heuristic way concepts of system complexity and cost. This discussion begins in the subsection (4) Some Heuristics About An Error Checking System (Section B) with a few brief comments.* The topic of cost and complexity continues in the examples of error-checking methods given in Sections D, F, and G. The importance of defining physically meaningful complexity measures is discussed in the subsection (5) Statement Of Research Methodology (Section E) and in the Summary (Section H).

The major topics developed in this paper are briefly summarized in Section H.

* "Error-checking" is a phrase used through out this paper to indicate a particular kind of system. Error-checking systems are those systems capable of any type of error detecting and/or correcting, i.e., capable of any type of "check" for errors.

B. A FIRST STEP TOWARDS AN ERROR CHECKING SYSTEM

(1) The Basic Mapping T

We are interested in performing a given mathematical mapping (operation). In this chapter the mapping to be performed will always be denoted by:

$$T \triangleq \text{the given mapping.}$$

The domain and range of T is always a finite set of nonnegative integers $\mathfrak{X}$:

$$\text{Domain } T = \text{Range } T = \mathfrak{X} = \{0,1,2,\ldots,M-1\}$$

where M-1 is the maximum element of $\mathfrak{X}$. In the remainder of this section (and paper) we will take the viewpoint that the mapping T accurately describes the input/output relationship of some physical system designed to realize the mapping T. The symbol T will serve double duty: (i) it will stand for the mathematical mapping in a rigorous sense and (ii) it will represent the physical system designed to perform the mapping T. Which usage is intended will be clear from context.

(2) The Decomposition of T

In order to take advantage of the desireable properties of residue number systems, the operation T maybe decomposed into the 3-stage process:

$$x \xrightarrow{En} r(x) \xrightarrow{OpT} t(x) \xrightarrow{De} T(x). \qquad (2.1)$$

The brief discussion of this process that follows is not intended to be mathematically rigorous. Its purpose is only to give a quick overview

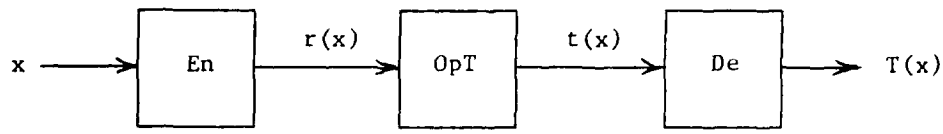


FIG. 37: BLOCK DIAGRAM (SYSTEM) REPRESENTATION FOR THE 3-STAGE PROCESS IN SECTION II. (2.1).

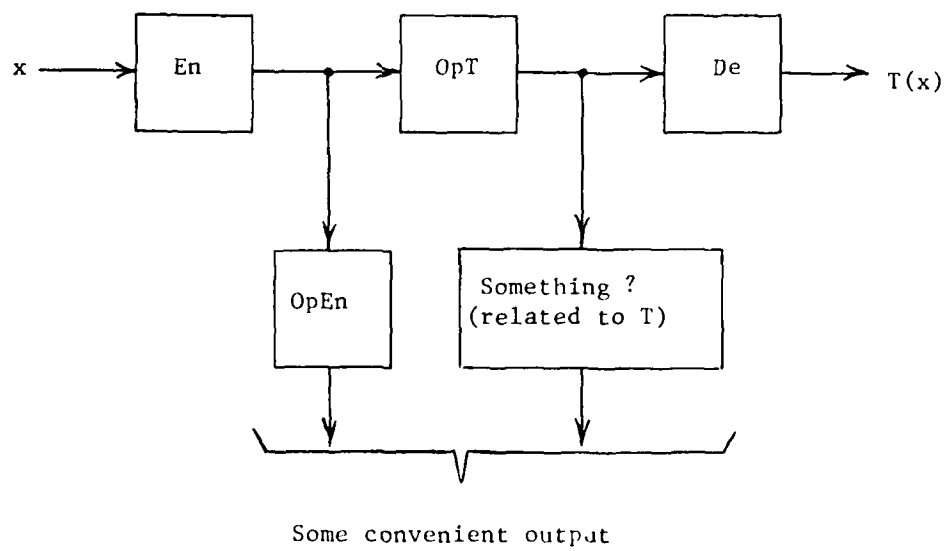


FIG. 38: THE FIRST STEP TOWARDS DEVELOPING AN ERROR CHECKING SYSTEM

of the process. Definitions and descriptions that need to be strengthened and made rigorous are deferred to the remaining sections.

First, the operation En "encodes" the integer input x into a nx1 "residue vector" r(x). This residue vector r(x) is just an ordered collection of elements whose form is $x \bmod m_i$ for i=1,2,...,n.*

Definition for r(x)

The residue vector r(x) is the integer input x encoded by the operation En:

$$x \xrightarrow{En} r(x) = \begin{vmatrix} x \bmod m_1 \\ x \bmod m_2 \\ \vdots \\ x \bmod m_n \end{vmatrix} \qquad (2.2)$$

where the set of moduli $\mathfrak{M} = \{m_i$ such that i=1,2,...,n$\}$ are pairwise relatively prime and the range M of the set $\mathfrak{M}$ is

$$M = \prod_{i=1}^{n} m_i .$$

After the input x has been encoded to the residue vector r(x), the operation OpT performs an operation on the residue vector r(x) equivalent to T's operation on x. The operation OpT is equivalent in the sense that T(x) can be determinded from the intermediate output t(x)=OpT(r(x)).

* A "residue vector" is any vector of the form used to define the residue vector r(x). Note that because the elements of the moduli set $\mathfrak{M}$ are all relatively prime by definition, there is no "redundant" information (with respect to mod operations) about the scalar input x in the residue vector r(x).

Definition for t(x)

The operation OpT maps the residue vector r(x) to the residue vector t(x):

$$r(x) \xrightarrow{OpT} = t(x) = \begin{vmatrix} T(x) \bmod m_1 \\ T(x) \bmod m_2 \\ \vdots \\ T(x) \bmod m_n \end{vmatrix}. \qquad (2.3)$$

This mapping represents the mathematical operation to be performed, i.e., addition, subtraction, etc.

In the third and final stage the operation De "decodes" the residue vector t(x) to the value T(x).

It is significant to note that any mapping defined on an integer set $\mathfrak{I} = \{0,1,2,\ldots,M-1\}$ to the same set can be realized by this decomposition. In essence, then, the first purpose of this paper is half completed: the 3-stage process described in (2.1) can characterize any mathematical mapping (operation) performed in a residue number system with range M.

But what about the errors? Why, how, and where do they arise? What is there nature? The "why and how" is a topic not treated in this chapter. The "where" and what to do about it is the primary topic of interest.

(3) On Noisy Residue Operations

First, consider a "noisy" encoding process En:

$$x \xrightarrow{\text{En}} r(x) + e(x) = r'(x) \qquad (2.4)$$

where $e(x)$ is a $n \times 1$ random vector. More will said about the nature of $e(x)$ in the following sections. Suffice it to say now that the form of the noisy residue vector $r'(x)$ remains suitable for input to the OpT operation, i.e., $r'(x)$ is itself a residue vector. In addition to a noisy encoding process En, the OpT and De operations might be noisy. More will be said about these cases in the following sections.

With the introduction of possible error in the encoding process we come to a key issue: How will a nonzero error vector $e(x)$ affect the output of the 3-stage process described in (2.1)? Without going into great detail at this point, it is possible to demonstrate that in general the error in the output is "not well behaved." As an illustration of what "not well behaved" means in a heuristic sense consider the following example.

Example.

The process below encodes with a noisy encoder En and then decodes with a perfect decoder De:

$$x \xrightarrow{\text{En}} r(x) + e(x) \xrightarrow{\text{De}} x + \text{De}(e(x)). \qquad (2.5)$$

The decoding operation can be realized as

$$y = \left[\sum_{i=1}^{n} \frac{M}{m_i} \cdot b_i \cdot (y \bmod m_i + \text{error in } \underline{\text{ith}} \text{ residue})\right] \bmod M \qquad (2.6)$$

where the b_i, $i=1$ to n, are a set of integers and all m_i are contained in the set of moduli used to encode x.

Suppose the input $x = y = 0$ in (2.5), then using the decoding formula in (2.6) the the decoded noisy output is given by

$$De(r'(x)) = x + De(e(x)) = 0 + De(e(x)) =$$

$$= [\sum_{i=1}^{n} \frac{M}{m_i} \cdot b_i \cdot (0 + \text{error in the } \underline{\text{ith}} \text{ residue})] \bmod M$$

$$= [\sum_{i=1}^{n} \frac{M}{m_i} \cdot b_i \cdot (\text{ error in the ith residue })] \bmod M. \qquad (2.7)$$

The result displayed in (2.7) demonstrates clearly that if just one of the residue elements $x \bmod m_i$ is in error by only 1 the decoded number is off by a multiple of the factor $(\frac{M}{m_i} \bmod m_i) \bmod M$! Hence a small error in the encoding process (the residue $x \bmod m_i$ is off by 1) can lead to large error in the output (the result is off by some multiple of the factor $(\frac{M}{m_i} \bmod m_i) \bmod M$). In principle, errors in the operation OpT will exhibit the same type of behavior as that discussed above.

In what ways may errors like those described in the above example be detected? In what ways may such errors be corrected? Sections C, D, and E, develop a mathematical system whose structure will allow meaningful and definite answers to these questions.

(4) Some Heuristics About An Error Checking System

It is interesting to note that even before one undertakes the developement of any particular system one can deduce some important facts about the general nature of such a system. First, we expect the "cost" of any system to be proportional to (i) the probability of error, (ii) the nature of the possible error and (iii) the overall complexity of the physical system. Secondly, any physically meaningful cost criterion will certainly be based on some measure of system complexity. This measure of complexity will, in general, take into account difficulties in (i) realizing given types of mathematical operations in a

physical system and (ii) interfacing the necessary operations. Finally, at this point, it seems reasonable to expect there will be *many* different conceptual approaches to realize error-checking. These different conceptual approaches will demand at least some differences in the physical systems designed to realize these approaches. Therefore, what we mean by system complexity will probably have a significant impact on the "costs" of different approaches.

C. Second Step Towards An Error Checking System

(1) The Basic Process

Shown in fig. 3.1 is a block diagram representation for the 3-stage process

$$x \xrightarrow{En} r(x) \xrightarrow{OpT} t(x) \xrightarrow{De} T(x) \qquad (2.1)$$

discussed in Section B. The scalar input x is encoded to a residue vector r(x). The residue vector r(x) is then transformed by OpT to the residue vector t(x). The vector t(x) is decoded by De to the output T(x). The basic process in fig. 3.1 has no explicit "error-checking" capability.* So, if any of the operations are error prone there is no way to check errors. It is possible, however, to develop a system that is derived from this basic system and is capable of error-checking.

(2) Introducing Error Checking Operations

As a first step towards developing this error-checking system, two operations will be added to the basic process. The result is the error-checking basic process in fig. 3.** Can it be this simple? Can we expect to accomplish error-checking simply by adding on the operations OpEn and "something?(related to T)."

The answer, clearly, is no. As given in the fig. 3.2 the new operations "look" only at the noisy residue vector r(x)+e(x) and the

---

* The phrase " error-checking" is used through out the remainder of the paper in place of the longer phrase "error detecting/correcting."

** The decoding operation De will be assumed to be error free through out the remainder of the paper.

noisy residue vector $t(x)+e'(x)$.* The way the encoding process En was defined in (3.2) does not allow for any redundancy in the residue (mod) encoding. The only "information" OpEn has about x is the noisy residue vector $r(x)+e(x)$. Similiarly, the only information the operation "something?(related to T)" has about $T(x)$ is the noisy residue vector $t(x)+e'(x)$. In order for the operation OpEn to check errors in the encoding of x to a residue vector, there must be some input to OpEn other than just the noisy residue vector $r(x)+e(x)$.**

(3) Introducing Pre-encoding

This leads us naturally to the concept of "pre-encoding." Pre-encoding is discussed in the beginning of Section D and again in Section E. In order for an error-checking operation OpEn to accomplish any error-checking, the operation OpEn "needs" more information about x than just the noisy residue vector $r(x)+e(x)$. The purpose of a pre-encoding operation is to generate different and independent information about x. This new information is used as additional input to some modified error-checking operation $\widetilde{\text{OpEn}}$. The new information is assumed to be error free.

* The vector $e'(x)$ is a random vector whose affect on $t(x)$ is analogous to the affect the random vector $e(x)$ has on $r(x)$ (discussed in Section B).

** In principle, the general description and structure of the error-checking operation OpEn given in Section E is sufficient to define a suitable error-checking operation OpT. In order to economize on words and simplify the structure of the paper, attention is focused mainly on OpEn. This does not imply a lack of generality.

D. Final Step Towards An Error Checking System

(1) The Basic System

The basic structure of the system illustrated in fig 4.1 is almost the same as the structure of the simple system developed in Section B and shown in fig. 3.1. In comparing the two systems, one can see that the only real difference in the structure is the addition of the operation $\widetilde{\text{PreEn}}$ which pre-encodes the input x.

In place of the integer input x we now have a vector $\tilde{x}$:

$$x \xrightarrow{\widetilde{\text{PreEn}}} \tilde{x}.$$

The vector $\tilde{x}$ should be thought of as a kx1 vector which "carries" information about the number x. The dimension k and the specific form of the individual elements in $\tilde{x}$ is not discussed again until Section E (in (2) Towards Defining $\tilde{x}$). The kind of information $\tilde{x}$ may carry about x is quite general: Whatever one might dream up and be able to encode. For example, $\tilde{x}$ maybe the input x repeated, some function of x, number theoretic properties like evenness/oddness or primeness. Following the pre-encoding step the vector $\tilde{x}$ passes through the remaining stages in a way analogous to the integer input x passing through the stages of the process shown in fig. 3.1.

(2) Description Of Basic System

First, $\tilde{x}$ is encoded by the encoding process $\widetilde{\text{En}}$:

$$\tilde{x} \xrightarrow{\widetilde{\text{En}}} \tilde{r}(x)$$

where $\tilde{r}(x)$ is some vector that takes the place of the residue vector

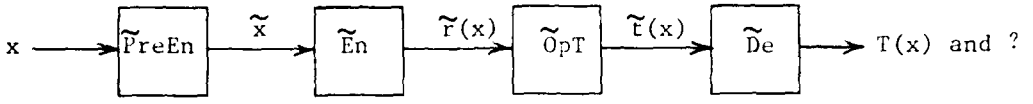


FIG. 39: THE BASIC SYSTEM USED TO CONSTRUCT AN ERROR CHECKING SYSTEM SHOWN IN FIG. 4.2.

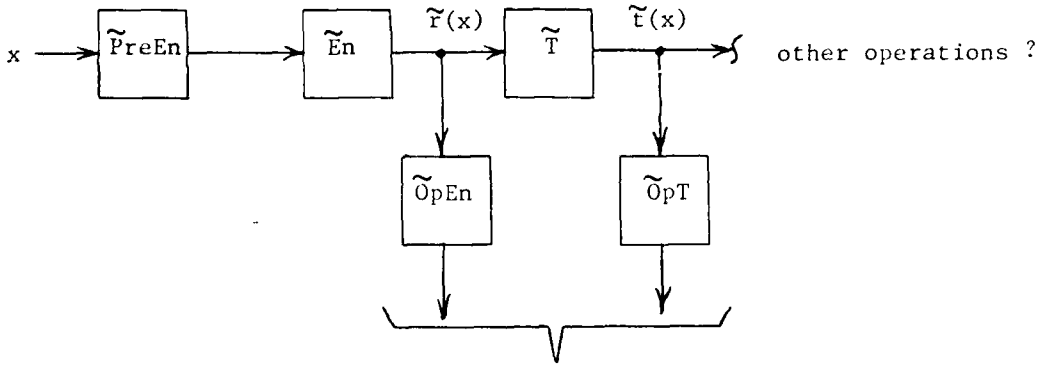


FIG. 40: THE ERROR CHECKING SYSTEM

r(x). In place of the original residue encoding process En is the encoding process En that encodes in part by mod operations and in part by a set operations that are necessary for error correction/detection. The vector $\tilde{r}(x)$ contains all the information about x available to the system. It passes through the operation $\tilde{O}pT$:

$$\tilde{r}(x) \xrightarrow{\tilde{O}pT} \tilde{t}(x).$$

The vector $\tilde{t}(x)$ contains all the information available to the system about x and T(x). Finally $\tilde{t}(x)$ is decoded by the decoder $\tilde{D}e$ to T(x):

$$\tilde{t}(x) \xrightarrow{\tilde{D}e} T(x).$$

It should be noted that the output of the decoder will in general really be some vector whose elements contain T(x),x and any other information in the pre-encoded $\tilde{x}$ vector. This then is the nature of the basic system. What will be added to this system for error detecting/correcting? What purpose does the pre-encoder $\tilde{P}reEn$ really serve? These questions are handled next.

(3) The Error Checking System

In fig 4.2 is the error-checking system. One can see that the basic system has been altered by eliminating any explicit mention of a decoding stage, relabelling the $\tilde{O}pT$ stage to $\tilde{T}$, and adding two new error checking operations $\tilde{O}pEn$ and $\tilde{O}pT$.* What are these error-checking operations $\tilde{O}pEn$ and $\tilde{O}pT$? How do they relate to the still mysterious pre-encoding operation $\tilde{P}reEn$?

* The decoding process $\tilde{D}e$ is not explictity shown for the sake of simplicity. The $\tilde{T}$ operation is, in principle, identical with the $\tilde{O}pT$ operation in fig. 4.1. The label is changed only to facilitate other notation.

Discussion of $\widetilde{O}pEn$ and $\widetilde{P}reEn$

The operation $\widetilde{O}pEn$ has as its input the vector $\tilde{r}(x)$. This vector hopefully contains the correct encoding of the input vector x. The operation OpEn is constructed so that any "inconsistencies" in the encoded information about x in $\tilde{r}(x)$ can be detected by examination of $\widetilde{O}pEn$'s output. Note that in general, the pre-encoding process $\widetilde{P}reEn$ and the error-checking process $\widetilde{O}pEn$ are highly related. Any particular choice for a pre-encoder certainly will be influenced by ideas about what types of operations $\widetilde{O}pEn$ are (i) mathematically interesting, (ii) conceptually powerful, (iii) physically realizeable and (iv) feasible under some cost criterion. Perhaps it will be instructive to give two straight-forward examples of error-checking operations. The nature of $\widetilde{P}reEn$ and $\widetilde{O}pEn$ will be described in each case.

Example one

This example shows "odd shift" errors can easily be detected. What are odd shift errors?

Definition for odd shift errors

Suppose a scalar input x is encoded by a noisy encoding process to the residue vector $r(x)+e(x)$ and then decoded to the scalar $x+De(e(x))$. If $De(e(x))$ is an odd number then by definition $r(x)$ (or equivalently x) has undergone an "odd shift" error in encoding. How can such errors be detected? If the set of moduli used to encode x are all odd, such detection can be accomplished by generating the additional information x mod2.

As a special case consider encoding the $\mathcal{I}$ domain $=\{0,1,2,\ldots,8\}$ by using only one modulus = 9. The $\mathcal{I}$ domain, x mod9 and x mod2 are depicted below in Table III.1.

x :	0	1	2	3	4	5	6	7	8	9	10	. . .
x mod9 :	0	1	2	3	4	5	6	7	8	9	10	. . .
x mod2 :	0	1	0	1	0	1	0	1	0	1	0	. . .

Table III.1

What happens if x mod9 undergoes an odd shift error? Without loss of generality suppose x were 0. Then x should encode to 0 mod9 and 0 mod2. Suppose however x mod9 encodes to one. We now have a residue vector

$$\tilde{r}(x) = \begin{vmatrix} x \bmod 9 = 1 \\ x \bmod 2 = 0 \end{vmatrix}.$$

But upon examination of Table III.1 we see that this residue vector does not correspond to any input value in the input domain $\mathcal{I}$. Therefore the information about x in $\tilde{r}(x)$ is inconsistent.* The example used was x mod9=1. By examining Table III.1 one can see that the information in the residue vector would be inconsistent if x were encoded to x mod9=1,3,5, or 7. One can also convince oneself by using examples of one's own that all oddshift errors for any $x \in \mathcal{I}$ input can be detected.

The pre-encoding process involved encoding x mod2. The error-checking operation $\tilde{O}pE_n$ decides if the information in $\tilde{r}(x)$ (in this case $\tilde{r}(x)$ is still a residue vector) is consistent. This can be accomplished by using a polynomial transformation with 2x9=18 degrees of freedom

* Notice that this $\tilde{r}(x)$ does correspond to the integer $9 \in \mathcal{I}$. That is, if we had encoded the $\mathcal{I}$ domain $= \{0,1,2,\ldots,17\}$ with the moduli set 2 and 9 then the $\tilde{r}(x)$ would not be inconsistent.

(DOF).* The increase in system cost for this type of error-checking should be expressible as something with the form:

Cost of error-checking + cost(x mod2 operation) + cost($\widetilde{O}$pEn with 18 DOF).

System cost is also decreased, however, due to the elimination of certain error types. The remaining cost due to errors has the form:

Cost after error-checking = cost(even shift errors).**

It is not necessary to restrict the residue encoding process to 1 modulus. Any set of relatively prime odd moduli used for encoding could be checked for odd shift errors using a perfect x mod2 encoding. The increase in system cost would be of the form:

Cost of error-checking = cost(x mod2 operation)
+ cost($\widetilde{O}$pEn with 2 M DOF).

Example two

This example demonstrates that all errors except "multiples of a particular type of redundant modulus" can be detected easily. What form do these redundant moduli take? Usually when one says two moduli m_1 and m_2 are redundant one means that m_1 and m_2 have common divisors. The type of redundant moduli used here are those for which one divides the other, e.g., m_1=3 and m_2=9.

As a special case consider the same $\mathfrak{X}$ domain as in example 1 and

---

* There are 2x9=18 possible inputs to $\widetilde{O}$pEn.
** I am assuming the nature of the error process is such that errors are more or less uniformly distributed over the input domain $\mathfrak{X}$.

encode x mod9 and x mod3. The $\mathfrak{X}$ domain, x mod9 and x mod3 are depicted in Table III.2 below.

x	:	0	1	2	3	4	5	6	7	8	9	10 . . .
x mod9	:	0	1	2	3	4	5	6	7	8	0	1 . . .
x mod3	:	0	1	2	0	1	2	0	1	2	0	1 . . .

Table III.2.

Now what happens when x mod9 is incorrectly encoded? Without loss of generality suppose x were 0. Then x should encode to 0 mod9 and 0 mod 3. If x mod9 encodes to 1,2,4,5,7, or 8 this will be inconsistent with the value for x mod3=0. Therefore error can be detected in these cases. Error can not be detected if x mod9=3 or 6.

As an aside, note that if the error process were such that the encoded residue x mod9 could shift from the correct value by only 1, then all errors could be detected and corrected using the additional information about x contained in the residue x mod3. This special case is discussed more fully and generalized in "Example: Special case of redundant encoding" (Section F).

The pre-encoding process involved encoding x mod3. The error-checking operation $\widetilde{O}pE_n$ decides if the information in x mod9 and x mod3 is consistent. This can be accomplished by using a polynomial transformation with 3x9=27 DOF. The increase in system cost for this type of error-checking takes the same form as that descibed in Example 1.

It is not necessary to restrict the residue encoding process to only one modulus. Many sets of relatively prime moduli used for encod-

ing could be checked for errors using an error-checking set of moduli with a smaller range. The technique requires only that the set with the smaller range be relatively prime and that the smaller range, call it S, divide the range of the set used for the encoding. The range of the encoding set was defined in Section B as M. Hence, S must divide M.* All errors in the $\mathcal{X}$ domain = {0,1,2,...M-1} can be detected except those of the form:

$$x \pm k \cdot S, \quad k=1,2,3,....$$

The increase in system cost should be of the form:

Cost of error-checking = cost(encode x with error-checking moduli set)
+ cost($\widetilde{O}$pEn operation with S M DOF).**

Discussion of $\widetilde{O}$pT

The error-checking operation $\widetilde{O}$pT is constructed, in principle, the same way the operation $\widetilde{O}$pEn is constructed. We know what operation T on x we want to perform. The nature of T and the design of the pre-encoder and $\widetilde{O}$pEn "designs for us" the necessay $\widetilde{T}$ operation. Based on the vector output $\widetilde{t}(x)$, the error-checking operation $\widetilde{O}$pT checks for any inconsistencies in the transformed information about x in $\widetilde{t}(x)$.

General Discussion

Note that in general, one would probably not construct any of the operations, $\widetilde{P}$reEn,$\widetilde{E}$n, $\widetilde{T}$, $\widetilde{O}$pEn, or $\widetilde{O}$pT without giving careful considera-

* The range S is the product of all the moduli in the error-checking set of moduli used to check the encoding.
** It turns out that the error correcting method discussed in "Example: Special case of redundant encoding" (Section F) has a significantly smaller cost than that suggested by this formula.

tion to the ways all the operations may interrelate. If some criterion for goodness of physical design takes into consideration range, speed, complexity, reliability, etc., we certainly expect these operations to be highly related.

For the sake of completeness a very brief discussion on how an overall system is controlled by outputs from error-checking operations is given next. Such a complete error-checking system is shown in fig. 4.3.

(4) Complete Error Checking System

In the error correcting system shown in fig 4.2 no provision was made either for the interpretation of the output from the error correcting operations, $\tilde{O}pEn$ and $\tilde{O}pT$, or for overall system control governed by such interpretation. For the sake of completeness the system in fig. 4.3 makes such provision by the addition of the System Self Checking Control Units.

The conceptual framework for a complete system has been developed. What do we do now? There are three related steps to take. First, develop as powerful and as orderly a method for investigating how the operations in the set $\{\tilde{P}reEn, \tilde{E}n, \tilde{O}pEn, \tilde{O}pT, \tilde{T}\}$ are related. Second, develop physically meaningful measures of system complexity, reliability and feasibility. Finally, use the knowledge and understanding gained in the first two steps to design actual physical systems. Section E undertakes the first step. Note is also taken of the importance of developing physically meaningful measures of complexity.

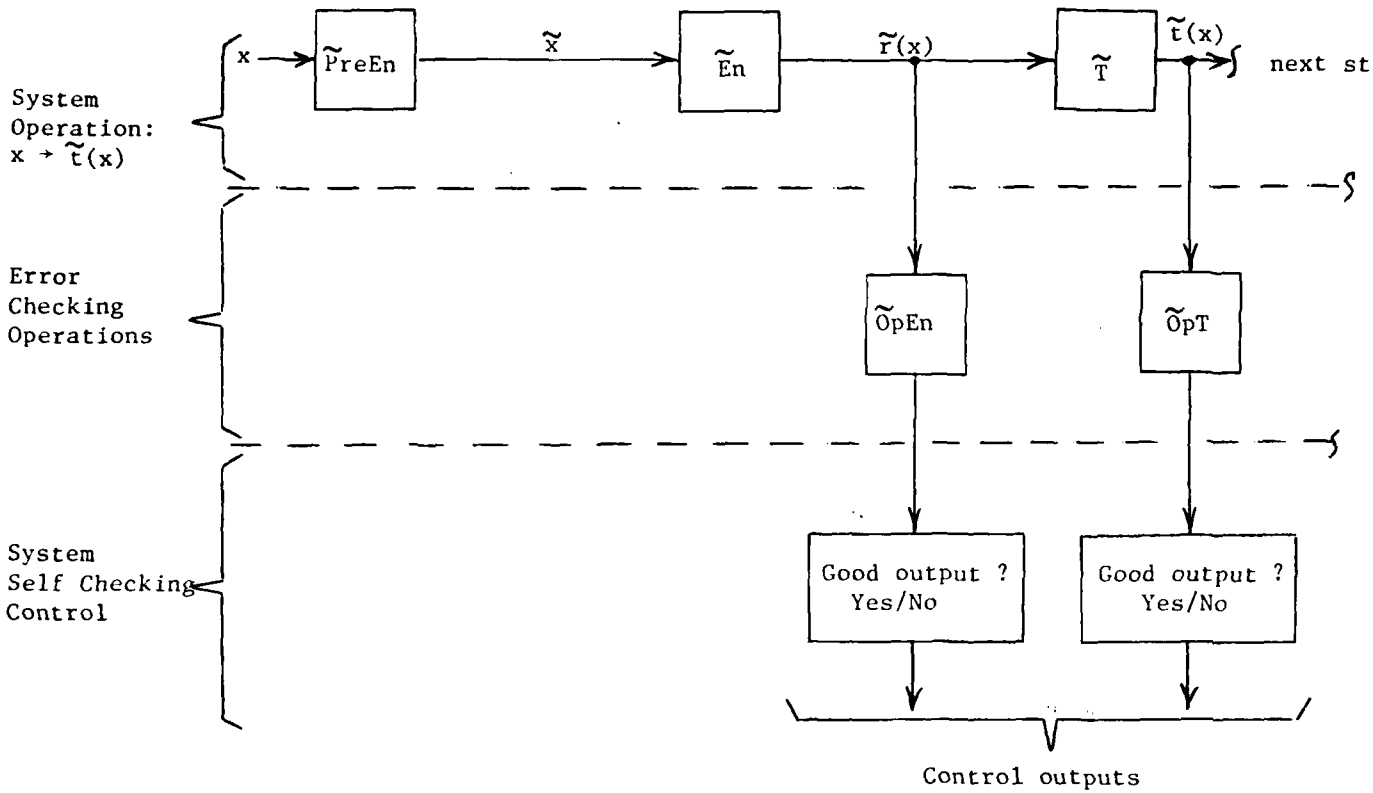


FIG. 41: THE COMPLETE ERROR CHECKING SYSTEM

E. RESEARCH METHODOLOGY

(1) General Discussion

A system concept general enough to treat most computational problems of interest as special cases is developed in Section D. A block diagram representation of this system is shown in Fig. 4.2. The purpose of this section is primarily to develope a methodology for research on error-checking methods. The system concept defined in Section D is only the first step towards this goal. We still need to impose additional mathematical structure on the system before a more rigorous mathematical analysis may begin. This mathematical structure will be imposed (implied) by the definitions for the various elements $\tilde{x}$, $\tilde{r}(x)$, and $\tilde{t}(x)$ as well as the operations $\tilde{P}reEn$, $\tilde{E}n$, $\tilde{O}pEn$, $\tilde{T}$, and $\tilde{O}pT$. There are, of course, an infinity of mathematical structures from which to choose.*

What attributes should the definitions have? Primarily we require the definitions to meet a "let's-make-progress" criterion. Secondly, the generality of the system must be maintained.** Thirdly, both the definitions themselves and the structure imposed on the system by the definitions must be clear enough so that the heuristic may serve as a powerful aide and guide to further mathematical analysis.

Fortunately, all three of these criteria can be met. There is a way of viewing the noisy residue encoding process

$$x \xrightarrow{Fn} r(x) + e(x) \tag{5.1}$$

---

* The author has considered a significant number there of!

** That is, we want to be able to realize any mapping from $\mathcal{A}$ to $\mathcal{A}$.

that implies a natural requirement on the vector $\tilde{x}$. Once $\tilde{x}$ is suitably defined, definitions for all the remaining elements and operations follow.

(2) Towards Defining $\tilde{x}$

Think of the scalar input x in (5.1) as representing the set of all possible incorrect outcomes of the noisy encoding process. This point of view establishes a correspondence between x and a set of residue vectors:

$$x \longleftrightarrow \{r(x) + e(x) \text{ for all } e(x) \text{ of possible interest }\}.^{*} \quad (5.2)$$

Now every r(x) + e(x) will decode to some number

$$r(x) + e(x) \xrightarrow{De} x + De(e(x)) = x + \hat{e}(x)$$

where $\hat{e}(x) = De(e(x))$. So, we might just as well establish the correspondence between x and the set of decoded residue vectors:

$$x \longrightarrow \&(x) = \{ x + \hat{e}(x) \text{ for all } \hat{e}(x)\}.^{**} \quad (5.3)$$

For the purposes of notational simplicity the latter correspondence is used in the discussion that follows.

As a next step the set &(x) in (4.3) is partitioned into the distinct sets

$$\&(x) = \{ x \} \cup \{ x + \hat{e}(x) \text{ for all } \hat{e}(x) \neq 0\}. \quad (5.4)$$

* In general, one presumedly would like a system to be error free. There may be special cases, however, where certain error types are considered far more costly or "disasterous" than others. In these cases the primary concern may be to eliminate only these particular error types.

** From here on the phrase " of possible interest" as a qualifier on the set will not be used explicitly. Unless stated otherwise, however, this qualifier is implied.

Define the set $\mathcal{P}(x)$ by

$$\mathcal{P}(x) = \{ x + \hat{e}(x) \text{ for all } \hat{e}(x) \neq 0 \}.$$

The set $\mathcal{P}(x)$, then, is the set containing all the possible outcomes of the noisy encoding process, given x is the input.

The input domain $\mathcal{I} = \{ 0,1,2,\ldots,M-1 \}$ can be partitioned then into the 2 distinct sets

$$\mathcal{I} = \mathcal{P}(x) \cup \mathcal{P}(x)^c.*$$

So we have a partition of $\mathcal{I}$? What purpose does it serve? We are looking for a suitably well defined definition for the vector $\tilde{x}$. If $\tilde{x}$ is to "supply" enough information about the true input such that all errors can be detected, then certainly it must supply enough information to tell us unambiguously if the output $x + \hat{e}(x)$ is an element of $\mathcal{P}(x)$ or $\mathcal{P}(x)^c$.

This error-checking may be accomplished if $\tilde{x}$ itself induces a partition on $\mathcal{X}$, the same partition on $\mathcal{I}$ induced by x. That is, x must contain at least enough information to induce the partition $\mathcal{P}(x) \cup \mathcal{P}(x)^c$. The error-checking can then be done by asking

$$\text{is } x + \hat{e}(x) \in \mathcal{P}(x) \text{ or not?}$$

Definition for $\tilde{x}$

The vector $\tilde{x}$ is the output of the $\widetilde{\text{PreEn}}$ pre-encoding process. It contains all the information about the scalar input x available to the system. It is a kx1 vector partitioned into the two subvec-

---

* $\mathcal{P}(x)^c$ means the compliment of $\mathcal{P}(x)$.

tors v(x) and a(x):

$$x \longrightarrow (v(x), a(x))$$

where v(x) is a ix1 vector and a(x) is a jx1 vector and k=i+j. The scalar input x is represented by the vector v(x); and the additional information about x is pre-encoded by the $\widetilde{\text{PreEn}}$ process to the vector a(x). The pre-encoding of

$$x \longrightarrow v(x)$$

is one-to-one and onto. The use of such coding allows for a general representation for the input x. Examples of such a general representation are:

(i) $x = x_1+x_2+x_3+\ldots+x_n \longrightarrow v(x) = (x_1,x_2,x_3,\ldots,x_n)$ and

(2) $x = x_1\ x_2\ x_3\ \ldots\ x_n \longrightarrow v(x) = (x_1,x_2,x_3,\ldots,x_n)$.

(3) Towards Defining $\widetilde{\text{PreEn}}$, $\widetilde{\text{En}}$, $\tilde{r}(\underline{x})$, $\widetilde{\text{OpEn}}$, $\widetilde{T}$, $\tilde{t}(\underline{x})$, $\widetilde{\text{OpT}}$

All of the operations $\widetilde{\text{PreEn}}$, $\widetilde{\text{OpEn}}$, $\widetilde{T}$ and $\widetilde{\text{OpT}}$ satisfy the following:

(i) the mappings have vector inputs (range) and vector outputs (domain),

(ii) the elements of any vector are nonnegative integers,

(iii) the number of possible vector inputs and outputs is finite, and

(iv) all of the mappings can be realized by polynomial transformations.

Definition for $\widetilde{\text{PreEn}}$

The pre-encoding process has been implicitly defined by the

definitions given for the scalar input x and the pre-encoded output vector $\tilde{x}$.

Definition for $\tilde{r}(\underline{x})$ and $\widetilde{En}(\underline{x})$.

The vector $\tilde{r}(x)$ is the ouput of the encoder $\widetilde{En}$:

$$\tilde{x} \xrightarrow{\widetilde{En}} \tilde{r}(x).$$

The vector $\tilde{r}(x)$ is a nx1 vector. In a specific case $\tilde{r}(x)$ might satisfy the equality:

$$\tilde{r}(x) = \widetilde{En}(x) = \widetilde{En}(\, v(x),\, a(x)\,) = (\, En_1(v(x)),\, En_2(a(x))$$

where the mappings En_1 and En_2 are independent operations acting independently on the vectors v(x) and a(x) as inputs; $En_1(.)$ is a 1x1 vector and $En_2(.)$ is a mx1 vector where 1+m=n. In general, however, the vector $\tilde{r}(x)$ will satisfy only the equality:

$$\tilde{r}(x) = \widetilde{En}(x).$$

Definition for $\widetilde{OpEn}$

The error-checking operation $\widetilde{OpEn}$ has as its input the nx1 vector $\tilde{r}(x)$ and a binary scalar output. The output is given by

$$\widetilde{OpEn}(\tilde{r}(x)) = \begin{cases} 0 \text{ if encoding inconsistent} \\ 1 \text{ if encoding consistent} \end{cases}$$

Definition for $\widetilde{T}$ and $\widetilde{t}(\underline{x})$

The operation $\widetilde{T}$ has as its input the nx1 vector $\tilde{r}(x)$ and a qx1 output vector $\widetilde{t}(x)$:

$$\tilde{r}(x) \xrightarrow{\tilde{T}} \left|\begin{array}{c} T_1(x) \\ \hline T_2(x) \end{array}\right| = t(x)$$

where $T_1(.)$ is ox1 vector, $T_2(.)$ is px1 vector and o+p=k. The subvector $T_1(x)$ is the residue vector for the scalar $T(x)$. The subvector $T_2(x)$ contains the additional information about x pre-encoded by $\tilde{P}reEn$.

Definition for $\tilde{O}pT$

The error-checking operation $\tilde{O}pT$ has as its input the kx1 vector $\tilde{t}(x)$ and a scalar binary output. The output is given by

$$\tilde{O}pT(\tilde{r}(x)) = \begin{cases} 0, \text{ if the mapping T is inconsitent} \\ 1, \text{ if the mapping T is consistent.} \end{cases}$$

(4) Towards Statement Of Research Methodology

The proposed research methodology is based primarily on the partitioning concept introduced in the subsection Towards Defining $\tilde{x}$. Any error detecting/correcting method, no matter how derived, implicitly induces a partition on the scalar input range . Therefore, obtaining a clear and thorough understanding of how partitions of the finite range can be generated from (i) a collection of numbers (the elements of a(x)) and (ii) a set of operations on these numbers, is a worthwhile goal.

If we had this understanding, the following questions could be more easily answered.

(i) Given a set of operations and a finite set of elements what classes of partitions can be parameterized by the set of elements?

AD-A107 568 STANFORD UNIV CA STANFORD ELECTRONICS LABS F/G 9/2
SYSTEM CONSIDERATIONS IN THE DESIGN OF RESIDUE PROCESSORS.(U)
MAR 79 A HUANG, J MANDEVILLE, J E GOODMAN AFOSR-77-3219
UNCLASSIFIED SU-SEL-79-008 AFOSR-TR-81-0744 NL

2 OF 2
AD A
107 568

(ii) For a given class of partitions that can be parameterized what tradeoffs exist between the number of elements used and the type, number and order of operations used to generate the partitions.

(iii) Is there a minimum cost approach under some cost criterion where the number of elements and the type, number, and order of operations used to generate a partition are the variables affecting cost.

(5) Statement Of Research Methodology

Our research efforts should be directed towards:

(i) acquiring a more thorough mathematical background on the nature of partitions and how to generate them,

(ii) continuing to generate partitions by ad hoc methods (see the next subsection entitled Some Adhoc Partitioning Methods),

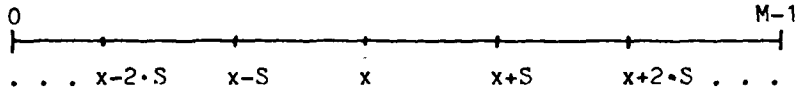
(iii) developing a variety of physically meaningful cost criteria (complexity measures) likely to suit most applications, and

(iv) investigating the relationship between the choice of a complexity measure and the resultant cost of a given partition method (a sensitivity analysis).

(6) Some Adhoc Partitioning Methods

(i) Number theoretic properties like evenness/oddness and the nature of the input number's prime factorization may be useful.

(ii) Partitions induced by encoding the input x mod some set of small moduli not subject to error may prove to be the best way. The partition of induced by this method given the input x is illustrated below:



where S = the product of the small error free moduli.

The partition induced on by the moding operations is

$$\{x\} \cup \{ (x+k\ S)\ \mathrm{mod} M,\ k = \text{nonzero integers} \}$$

(iii) Partitions may be induced by passing lower and upper bounds for the input x, i.e., $x \in (x-a, x+b)$.

(iv) Combine the above two methods.

(v) Partitions could be generated by carrying out operations on the residue vector. For example, a set of values $\{ (\tilde{r}(x), O_i(\tilde{r}(x)) \mid i=1,\ldots? \}$, where (.,.) is an inner product and O_i (.) is an operation defined on $\tilde{r}(x)$, could be computed.

F. ERROR CHECKING BASED ON ADDITIONAL ERROR FREE RESIDUE ENCODING

(1) General Discussion

We are looking for ways to check errors in residue encoding. It is a natural question to ask "Can we use residue encoding to check residue encoding?" The answer, in general, is yes. Simple examples of this kind of error-checking are given in Section D. All the examples in this section are also of this type. There is a basic form to this type of error-checking. The following discussion describes this form. The pre-encoding operation $\tilde{P}reEn$ generates a residue vector $\tilde{p}(x)$:

$$\tilde{p}(x) = \begin{vmatrix} x \bmod s_1 \\ x \bmod s_2 \\ \vdots \\ x \bmod s_n \end{vmatrix}$$

where $\mathcal{S} = \{s_i \mid i=1,2,\ldots,n\}$ is a set of error free moduli (generally small), all s_i are relatively prime, and the range of $\mathcal{S}$ is S. The encoding operation $\tilde{E}n$ generates a residue vector $\tilde{r}(x)$:

$$\tilde{r}(x) = \begin{vmatrix} x \bmod m_1 \\ x \bmod m_2 \\ \vdots \\ x \bmod m_n \end{vmatrix}$$

where $\mathcal{M} = \{m_i \mid i=1,2,\ldots,n\}$ is a set of noisy moduli (generally large), all m_i are relatively prime, and the range of $\mathcal{M}$ is M. The pre-encoded residue vector $\tilde{p}(x)$ then is used to check for errors in the residue vector $\tilde{r}(x)$.

For special kinds of error processes, the error-checking operations

based on this method might prove to be cost effective. The method discussed below in "Example: Special case of redundant encoding" is a good example of such a special case. It should be pointed out, however, that in order to detect (or correct) all possible errors for any error process, the range of the small moduli $\mathcal{S}$ set, S, must be at least as large as the range of the encoding set $\mathcal{M}$, M. If S=M, clearly one would encode the input x with the error free moduli set $\mathcal{S}$ and not use the error prone set $\mathcal{M}$ at all!

It is convenient to separate the method of using additional error free moduli into two distinct types: (i) the range S divides M and (ii) the range S does not divide M. These two distinct types are discussed in the two following subsections. The "S divides M" type is handled first.

(2) Discussion of "S divides M" Error Checking

Example 2 (Section D) gives an overall description of the nature, behavior, and cost of such error-checking systems. As indicated in Example two, the error-checking method is capable of detecting all errors except those of the form:

$$x \pm k \cdot S, \quad k = 1,2,....$$

The method can not in general correct all detectable errors. However, if the nature of the error process is of a certain type, it is possible, in principle, to detect and correct all errors. Example two takes note of this possibility for noisy mod9 encoding when the error can shift the correct residue value by at most 1. This special case uses only one modulus. This can be generalized to encoding with n noisy moduli and

error-checking with n error free moduli. The following example does just that.

Example: Special case of redundant encoding

There are three basic requirements that a given noisy residue encoding process must satisfy for this special case of error correcting to work. One requirement restricts the class of error processes possible. The other two requirements restrict the choice of moduli used. The requirements are:

(i) the error process for each $\text{mod}m_i$ encoding must satisfy the following:

$|\text{noisy x mod}m_i \text{ encoding} - \text{true x mod}m_i \text{ value}| = |\text{error}| \leq$

maximum residue shift from error = maxrs <

the modulus m_i for all possible error,

(ii) there must exist an error free modulus s_i for each m_i that satisfies:

$$s_i \geq 2 \cdot \text{maxrs} + 1,$$

(iii) and the error free modulus s_i must divide each m_i, the modulus that s_i checks.

Example 2 (Section D) gives an example satisfying the above requirements. The maximum residue shift from error is 1. The modulus $m_1 = 9$. The error checking modulus is $s_1 = 3$ (which obviously divides 9). The cost of correcting errors is:

Cost of correcting errors = cost(x mod3 encoding)

+ cost(OpEn operation with 3x9 = 27 DOF).

A generalization of this error correction procedure is straight forward. For every $m_i \in \mathcal{M} = \{m_i \text{ such that } i=1,2,\ldots,n\}$ = "noisy encoding set" there must be a corresponding $s_i = \{s_i \mid i=1,2,\ldots,n\}$ = "error-checking encoding set" such that the three above requirements are met. The error-checking operation $\tilde{O}pEn$ used to correct the noisy residue values x $\text{mod}m_i$ can be decomposed into n independent operations. There exists a one-to-one ocrrespondence between the "ith residue pair" defined by

$$\text{ith residue pair} \triangleq (x \text{mod} m_i + \text{error}, x \text{mod} s_i)$$

and the "ith independent error-checking operation."

This decomposition can be done because each modulus m_i (or s_i) supplies information about the input value x independent of all the other moduli in the set $\mathcal{M}$(or $\mathcal{S}$). The moduli pairs (m_i, s_i) are redundant however, and therefore x $\text{mod}s_i$ supplies information about x that is not independent of the information supplied by the x $\text{mod}m_i$ value.

The ith independent error correction operation in $\tilde{O}pEn$ requires

$$s_i \cdot m_i \text{ DOF.}$$

Therefore the cost of the ith operation is:

Cost of correction x $\text{mod}m_i$ error = cost(x $\text{mod}s_i$ encoding)

+ cost(operation with $s_i \cdot m_i$ DOF).

The total cost of error correcting then is:

Cost of error correcting r(x) = cost(encode x with δ)

$$+ \sum_{i=1}^{n} \text{cost(operation with } s_i \cdot m_i \text{ DOF).}$$

The following example illustrates the essentials of this method.

Example

Suppose encoding is done with the set $\mathcal{M} = \{121,169\}$. If the maximum residue shift from error is 5, then the set $\delta = \{11,13\}$ could be used to check for all errors in residue encoding with modulus set . The set δ could also be used to check for all errors in a general T operation if the maximum residue shift remains 5. The cost of error correcting is:

Cost of error correcting = cost(encode x with δ)
+ cost(operation with 11x121 DOF)
+ cost(operation with 13x169 DOF).

The nature of error processes intrinsic to analog encoding of residues often satisfy the requirements above. Therefore this special case of error correcting should prove to be useful in situations where analog processes are used to do mod encoding and residue arithmetic in general.

(3) Discussion Of "S does not divide M" Error Checking

Example one (Section D) is a special case of this type of error-checking. In the example, x mod2 encoding is used to detect all odd shift errors. One might hope that detecting even shift errors is just as simple. Unfortunately, it is not. If some small odd modulus $\overline{s}$ is used to check for even shift errors, it can not detect even shift errors

of the form:

$$\pm k \cdot 2 \cdot \bar{s}, \quad k=1,2,.....$$

where x is the input scalar and $\bar{s}$ is some small odd modulus assumed to be error free.

As an example, consider Table III.2 in Example two (Section D). Suppose the input were 0. If the small odd modulus were 3, the value x mod3 could not detect the error

$$x \bmod 9 + \text{error} = 0 + 6 = 0 + 2x3$$

At best, error-checking methods based on an error free set of moduli $\mathcal{J}$ = {2, and other relatively prime odd moduli} can detect all errors except those of the form

$$x \pm k \cdot S, \quad k=1,2,3,... \tag{6.1}$$

where S is the range of $\mathcal{J}$. In general these methods can <u>not</u> correct all detectable errors.

The cost of error-checking is:

$$\text{Cost of error-checking} = \text{cost(encode with } \mathcal{J} \text{ moduli set)} + \text{cost}(\tilde{O}\text{pEn operation with S}\cdot\text{M DOF}). \tag{6.2}$$

(4) <u>Final Discussion</u>

Special structure in the error process intrinsic to the mod encoding (or residue arithmetic operation) can lead to error-checking methods capable of detecting and even correcting all errors. The method described in "Example: Special case of redundant encoding" is a non-trivial example. In general, however, error-checking based on

generating additional information about x by performing extra error-free mod encoding can not do any better than that indicated in the results (6.1) and (6.2) (in the previous subsection).

G. SOME MORE ERROR CHECKING METHODS

(1) General Discussion

The discussion of the three main topics in this section is for the most part heuristic. First, methods of error-checking other than the use of additional error free moduli are discussed. Second, methods using the pre-encoding:

$$x \longrightarrow v(x)$$

as defined in the "Definition for $\tilde{x}$" (Section E) are briefly covered. Lastly, a way to perform the residue arithmetic operations

$$x \bmod m_i \longrightarrow T(x) \bmod m_i,\ i=1,2,\ldots,n \qquad (7.1)$$

using a set of small moduli whose range is $\geq m_i$ for all $i=1,2,\ldots,n$ is discussed.

(2) Error-Checking Without Additional Error-Free Residue Encoding

The partitioning methods given in (i), (iii), (iv), and (v) in (6) Some Adhoc Partitioning Methods (Section E) are examples of such methods. Is it possible that these methods can do a substantially better job of error-checking than methods using additional error free residue encoding for the same cost? This appears doubtful. The following discussion indicates why this is so.

A given error-checking operation for some residue arithmetic operation T,

$$r(x) \xrightarrow{T} t(x)$$

must generate numbers other than residue values from the input (x), r(x), and the output t(x). The numbers are then compared. If there are M possible inputs (and outputs) and the number of possible errors for any given input is N then any error-checking operation has, in general,

$$M \cdot N \text{ DOF.}$$

However, there appear to be two reasons why such methods may prove cost effective in some situations:

(i) error-checking operations which generate partitions different from those that can be easily generated with residue encoding may be necessary and

(ii) ordinary scalar arithmetic operations on residue $x \bmod m_i$ values may prove to be easier (less costly) to perform than residue encoding.

(3) Alternate x Input Representations

In "Definition for $\tilde{x}$" (Section E) mention is made of pre-encoding the scalar input x to some alternate representation:

$$x \longrightarrow (x_1, x_2, x_3, \ldots, x_n) \text{ where } x = x_1\ x_2\ x_3 \cdots x_n.$$

Why do this at all? Suppose the input domain $\mathcal{I} = \{0,1,2,\ldots,M-1\}$. Then it might be possible to use a set of x_i such that each x_i satisfies

$$x_i \leq \sqrt[n]{M}\ .$$

If each element of v(x) is residue encoded separately, then the range of the residue encoding process is reduced to only $\sqrt[n]{M}$.

This reduction in the range necessary to encode x is significant

and certainly would help make systems designed to perform arithmetic operations based on a residue number system less susceptible to encoding error. Unfortunately, using the v(x) representation for x does destroy the nice structure for arithmetic operations inherent in a residue number system with the full range M. Hence, the cost of "doing business" , i.e., the cost of performing an actual arithmetic operation T will substantially increase.

(4) Small Residues To Do Large Residue Arithmetic Operations

For some residue arithmetic operation T each residue $x \bmod m_i$ is mapped to:

$$x \bmod m_i \longrightarrow T(x) \bmod m_i, \quad i=1,2,\ldots,n. \tag{7.1}$$

But what is the mapping in (7.1)? It is a mapping from the finite set $\{0,1,2,\ldots,m_i-1\}$ for all $i=1,2,\ldots,n$ to itself. Therefore residue arithmetic could be used to accomplish it!

Suppose we have a moduli set $Q = \{q_i \mid i=1,2,\ldots,k\}$ where the range of Q is Q and $Q \geq m_i$ for all $i=1,2,\ldots,n$. Then this one set of small moduli could used to compute all the mappings in (7.1)!

The cost of doing each residue arithmetic operation directly in (7.1) is:

$$\text{Cost of ith residue operation} = \text{cost(operation with } m_i \text{ DOF)} + \text{cost(error with } m_i\text{)}.$$

The cost of doing each residue operation with the set Q is:

$$\text{Cost of ith residue operation} = [\ \text{cost(encode } x \bmod m_i \text{ with } Q)\ ]$$

$$+ \sum_{i=1}^{k} \text{cost(operation with } q_i \text{ DOF)}$$

$$+ \text{ cost(error with } Q)$$

$$+ [\ \text{cost(Decode to } T(x) \bmod m_i)\ ].$$

The cost of encoding and decoding with Q are in brackets because it is possible to design a system that performs many different T mappings such that the encoding with Q is done only once before all the T mappings and the decoding is done only once after all the T operations.

If it is intrinsically easier to generate x $\bmod m_i$ residue encoding for large m_i than it is to generate the arbitrary T mappings in (7.1), this method may prove particularly cost effective.

H. SUMMARY

A general system concept able to incorporate all residue arithmetic operations and error correcting/detecting methods as special cases is developed. The block diagram representation in Fig. 4.3 depicts this system.

A methodology for research into the intrinsic capabilities of error correcting/detecting methods is proposed. The motivation, main development, and statement of this methodology is given in Section E.

Several simple examples of error correcting/detecting methods are given. In particular, the method explained in "Example: Special case of redundant coding" might prove to be cost effective when analog processes involving thresholding are used to perform residue arithmetic operations.

The topic of system complexity and resultant cost is an important one. The need to develop physically meaningful measures of system complexity is noted in (5) Statement of Research Methodology (Section E). The costs of various error-checking methods are discussed in the examples given. Upon comparing the form of the costs as given, it is clear that more quantitative statements need to be made in order to chose with confidence between some of the different methods.

APPENDIX

ADDRESS	CONT	ADDRESS	CONT	ADDRESS	CONT
000000	X000	010101	XXXX	101010	XXXX
000001	X001	010110	XXXX	101011	XXXX
000010	X010	010111	XXXX	101100	XXXX
000011	X011	011000	X011	101101	XXXX
000100	X100	011001	X100	101110	XXXX
000101	XXXX	011010	X000	101111	XXXX
000110	XXXX	011011	X001	110000	XXXX
000111	XXXX	011100	X010	110001	XXXX
001000	X001	011101	XXXX	110010	XXXX
001001	X010	011110	XXXX	110011	XXXX
001010	X011	011111	XXXX	110100	XXXX
001011	X100	100000	X100	110101	XXXX
001100	X000	100001	X000	110110	XXXX
001101	XXXX	100010	X001	110111	XXXX
001110	XXXX	100011	X010	111000	XXXX
001111	XXXX	100100	X011	111001	XXXX
010000	X010	100101	XXXX	111010	XXXX
010001	X011	100110	XXXX	111011	XXXX
010010	X100	100111	XXXX	111100	XXXX
010011	X000	101000	XXXX	111101	XXXX
010100	X001	101001	XXXX	111110	XXXX
				111111	XXXX

TABLE T1

ADDRESS	CONT
000000	X000
000001	X001
000010	X010
000011	X011
000100	X100
000101	X101
000110	X110
000111	XXXX
001000	X001
001001	X010
001010	X011
001011	X100
001100	X101
001101	X110
001110	X000
001111	XXXX
010000	X010
010001	X011
010010	X100
010011	X101
010100	X110

ADDRESS	CONT
010101	X000
010110	X001
010111	XXXX
011000	X011
011001	X100
011010	X101
011011	X110
011100	X000
011101	X001
011110	X010
011111	XXXX
100000	X100
100001	X101
100010	X110
100011	X000
100100	X001
100101	X010
100110	X011
100111	XXXX
101000	X101
101001	X110

ADDRESS	CONT
101010	X000
101011	X001
101100	X010
101101	X011
101110	X100
101111	XXXX
110000	X110
110001	X000
110010	X001
110011	X010
110100	X011
110101	X100
110110	X101
110111	XXXX
111000	XXXX
111001	XXXX
111010	XXXX
111011	XXXX
111100	XXXX
111101	XXXX
111110	XXXX
111111	XXXX

TABLE T2

ADDRESS	CONT
000000	X000
000001	X001
000010	X010
000011	X011
000100	X100
000101	X101
000110	X110
000111	X111
001000	X001
001001	X010
001010	X011
001011	X100
001100	X101
001101	X110
001110	X111
001111	X000
010000	X010
010001	X011
010010	X100
010011	X101
010100	X110

ADDRESS	CONT
010101	X111
010110	X000
010111	X001
011000	X011
011001	X100
011010	X101
011011	X110
011100	X111
011101	X000
011110	X001
011111	X010
100000	X100
100001	X101
100010	X110
100011	X111
100100	X000
100101	X001
100110	X010
100111	X011
101000	X101
101001	X110

ADDRESS	CONT
101010	X111
101011	X000
101100	X001
101101	X010
101110	X011
101111	X100
110000	X110
110001	X111
110010	X000
110011	X001
110100	X010
110101	X011
110110	X100
110111	X101
111000	X111
111001	X000
111010	X001
111011	X010
111100	X011
111101	X100
111110	X101
111111	X110

TABLE T3

ADDRESS	CONT	ADDRESS	CONT	ADDRESS	CONT
000000	X000	010101	XXXX	101010	XXXX
000001	X001	010110	XXXX	101011	XXXX
000010	X010	010111	XXXX	101100	XXXX
000011	X011	011000	XXXX	101101	XXXX
000100	X100	011001	XXXX	101110	XXXX
000101	X101	011010	XXXX	101111	XXXX
000110	X110	011011	XXXX	110000	XXXX
000111	X111	011100	XXXX	110001	XXXX
001000	XXXX	011101	XXXX	110010	XXXX
001001	XXXX	011110	XXXX	110011	XXXX
001010	XXXX	011111	XXXX	110100	XXXX
001011	XXXX	100000	XXXX	110101	XXXX
001100	XXXX	100001	XXXX	110110	XXXX
001101	XXXX	100010	XXXX	110111	XXXX
001110	XXXX	100011	XXXX	111000	XXXX
001111	XXXX	100100	XXXX	111001	XXXX
010000	XXXX	100101	XXXX	111010	XXXX
010001	XXXX	100110	XXXX	111011	XXXX
010010	XXXX	100111	XXXX	111100	XXXX
010011	XXXX	101000	XXXX	111101	XXXX
010100	XXXX	101001	XXXX	111110	XXXX
				111111	XXXX

TABLE T4

ADDRESS	CONT	ADDRESS	CONT	ADDRESS	CONT
000000	X000	010101	X001	101010	X011
000001	X001	010110	X010	101011	X100
000010	X010	010111	X011	101100	X000
000011	X011	011000	X100	101101	X001
000100	X100	011001	X000	101110	X010
000101	X000	011010	X001	101111	X011
000110	X001	011011	X010	110000	X100
000111	X010	011100	X011	110001	X000
001000	X011	011101	X100	110010	X001
001001	X100	011110	X000	110011	X010
001010	X000	011111	X001	110100	X011
001011	X001	100000	X011	110101	X100
001100	X010	100001	X100	110110	X000
001101	X011	100010	X000	110111	X001
001110	X100	100011	X001	111000	X010
001111	X000	100100	X010	111001	X011
010000	X001	100101	X011	111010	X100
010001	X010	100110	X100	111011	X000
010010	X011	100111	X000	111100	X001
010011	X100	101000	X001	111101	X010
010100	X000	101001	X010	111110	X011
				111111	X100

TABLE T5

ADDRESS	CONT
000000	X000
000001	X001
000010	X010
000011	X011
000100	X100
000101	X101
000110	X110
000111	X000
001000	X001
001001	X010
001010	X011
001011	X100
001100	X101
001101	X110
001110	X000
001111	X001
010000	X010
010001	X011
010010	X100
010011	X101
010100	X110

ADDRESS	CONT
010101	X000
010110	X001
010111	X010
011000	X011
011001	X100
011010	X101
011011	X110
011100	X000
011101	X001
011110	X010
011111	X011
100000	X011
100001	X100
100010	X101
100011	X110
100100	X000
100101	X001
100110	X010
100111	X011
101000	X100
101001	X101

ADDRESS	CONT
101010	X110
101011	X000
101100	X001
101101	X010
101110	X011
101111	X100
110000	X101
110001	X110
110010	X000
110011	X001
110100	X010
110101	X011
110110	X100
110111	X101
111000	X110
111001	X000
111010	X001
111011	X010
111100	X011
111101	X100
111110	X101
111111	X110

TABLE T6

ADDRESS	CONT
000000	X000
000001	X001
000010	X010
000011	X011
000100	X100
000101	X101
000110	X110
000111	X111
001000	X000
001001	X001
001010	X010
001011	X011
001100	X100
001101	X101
001110	X110
001111	X111
010000	X000
010001	X001
010010	X010
010011	X011
010100	X100

ADDRESS	CONT
010101	X101
010110	X110
010111	X111
011000	X000
011001	X001
011010	X010
011011	X011
011100	X100
011101	X101
011110	X110
011111	X111
100000	X000
100001	X001
100010	X010
100011	X011
100100	X100
100101	X101
100110	X110
100111	X111
101000	X000
101001	X001

ADDRESS	CONT
101010	X010
101011	X011
101100	X100
101101	X101
101110	X110
101111	X111
110000	X000
110001	X001
110010	X010
110011	X011
110100	X100
110101	X101
110110	X110
110111	X111
111000	X000
111001	X001
111010	X010
111011	X011
111100	X100
111101	X101
111110	X110
111111	X111

TABLE T7

ADDRESS	CONT
000000	X000
000001	X011
000010	X110
000011	X010
000100	X101
000101	X001
000110	X100
000111	XXXX
001000	X100
001001	X000
001010	X011
001011	X110
001100	X010
001101	X101
001110	X001
001111	XXXX
010000	X001
010001	X100
010010	X000
010011	X011
010100	X110

ADDRESS	CONT
010101	X010
010110	X101
010111	XXXX
011000	X101
011001	X001
011010	X100
011011	X000
011100	X011
011101	X110
011110	X010
011111	XXXX
100000	X010
100001	X101
100010	X001
100011	X100
100100	X000
100101	X011
100110	X110
100111	XXXX
101000	XXXX
101001	XXXX

ADDRESS	CONT
101010	XXXX
101011	XXXX
101100	XXXX
101101	XXXX
101110	XXXX
101111	XXXX
110000	XXXX
110001	XXXX
110010	XXXX
110011	XXXX
110100	XXXX
110101	XXXX
110110	XXXX
110111	XXXX
111000	XXXX
111001	XXXX
111010	XXXX
111011	XXXX
111100	XXXX
111101	XXXX
111110	XXXX
111111	XXXX

TABLE T8

ADDRESS	CONT
000000	X000
000001	X101
000010	X010
000011	X111
000100	X100
000101	X001
000110	X110
000111	X011
001000	X011
001001	X000
001010	X101
001011	X010
001100	X111
001101	X100
001110	X001
001111	X110
010000	X110
010001	X011
010010	X000
010011	X101
010100	X010

ADDRESS	CONT
010101	X111
010110	X100
010111	X001
011000	X001
011001	X110
011010	X011
011011	X000
011100	X101
011101	X010
011110	X111
011111	X100
100000	X100
100001	X001
100010	X110
100011	X011
100100	X000
100101	X101
100110	X010
100111	X111
101000	XXXX
101001	XXXX

ADDRESS	CONT
101010	XXXX
101011	XXXX
101100	XXXX
101101	XXXX
101110	XXXX
101111	XXXX
110000	XXXX
110001	XXXX
110010	XXXX
110011	XXXX
110100	XXXX
110101	XXXX
110110	XXXX
110111	XXXX
111000	XXXX
111001	XXXX
111010	XXXX
111011	XXXX
111100	XXXX
111101	XXXX
111110	XXXX
111111	XXXX

TABLE T9

ADDRESS	CONT	ADDRESS	CONT	ADDRESS	CONT
000000	X000	010101	X101	101010	X011
000001	X111	010110	X100	101011	X010
000010	X110	010111	X011	101100	X001
000011	X101	011000	X011	101101	X000
000100	X100	011001	X010	101110	X111
000101	X011	011010	X001	101111	X110
000110	X010	011011	X000	110000	X110
000111	X001	011100	X111	110001	X101
001000	X001	011101	X110	110010	X100
001001	X000	011110	X101	110011	X011
001010	X111	011111	X100	110100	X010
001011	X110	100000	X100	110101	X001
001100	X101	100001	X011	110110	X000
001101	X100	100010	X010	110111	X111
001110	X011	100011	X001	111000	XXXX
001111	X010	100100	X000	111001	XXXX
010000	X010	100101	X111	111010	XXXX
010001	X001	100110	X110	111011	XXXX
010010	X000	100111	X101	111100	XXXX
010011	X111	101000	X101	111101	XXXX
010100	X110	101001	X100	111110	XXXX
				111111	XXXX

TABLE T10

ADDRESS	CONT	ADDRESS	CONT	ADDRESS	CONT
000000	0000	010101	XXXX	101010	XXXX
000001	0001	010110	XXXX	101011	XXXX
000010	0010	010111	XXXX	101100	XXXX
000011	0011	011000	XXXX	101101	XXXX
000100	0100	011001	XXXX	101110	XXXX
000101	XXXX	011010	XXXX	101111	XXXX
000110	XXXX	011011	XXXX	110000	XXXX
000111	XXXX	011100	XXXX	110001	XXXX
001000	XXXX	011101	XXXX	110010	XXXX
001001	XXXX	011110	XXXX	110011	XXXX
001010	XXXX	011111	XXXX	110100	XXXX
001011	XXXX	100000	XXXX	110101	XXXX
001100	XXXX	100001	XXXX	110110	XXXX
001101	XXXX	100010	XXXX	110111	XXXX
001110	XXXX	100011	XXXX	111000	XXXX
001111	XXXX	100100	XXXX	111001	XXXX
010000	XXXX	100101	XXXX	111010	XXXX
010001	XXXX	100110	XXXX	111011	XXXX
010010	XXXX	100111	XXXX	111100	XXXX
010011	XXXX	101000	XXXX	111101	XXXX
010100	XXXX	101001	XXXX	111110	XXXX
				111111	XXXX

TABLE T11

ADDRESS	CONT	ADDRESS	CONT	ADDRESS	CONT
000000	0000	010101	1111	101010	0000
000001	0000	010110	1111	101011	0000
000010	0000	010111	1111	101100	1111
000011	0000	011000	0000	101101	1111
000100	1111	011001	0000	101110	1111
000101	1111	011010	0000	101111	1111
000110	1111	011011	0000	110000	0000
000111	1111	011100	1111	110001	0000
001000	0000	011101	1111	110010	0000
001001	0000	011110	1111	110011	0000
001010	0000	011111	1111	110100	1111
001011	0000	100000	0000	110101	1111
001100	1111	100001	0000	110110	1111
001101	1111	100010	0000	110111	1111
001110	1111	100011	0000	111000	XXXX
001111	1111	100100	1111	111001	XXXX
010000	0000	100101	1111	111010	XXXX
010001	0000	100110	1111	111011	XXXX
010010	0000	100111	1111	111100	XXXX
010011	0000	101000	0000	111101	XXXX
010100	1111	101001	0000	111110	XXXX
				111111	XXXX

TABLE T12

ADDRESS	CONT	ADDRESS	CONT	ADDRESS	CONT
000000	0000	010101	1010	101010	0101
000001	0010	010110	1100	101011	1000
000010	0100	010111	1110	101100	1000
000011	0110	011000	0000	101101	1011
000100	0111	011001	0011	101110	1101
000101	1001	011010	0101	101111	1111
000110	1011	011011	0111	110000	0001
000111	1101	011100	1000	110001	0100
001000	0000	011101	1010	110010	0110
001001	0010	011110	1100	110011	1000
001010	0100	011111	1110	110100	1001
001011	0110	100000	0001	110101	1011
001100	0111	100001	0011	110110	1101
001101	1001	100010	0101	110111	1111
001110	1011	100011	0111	111000	XXXX
001111	1110	100100	1000	111001	XXXX
010000	0000	100101	1010	111010	XXXX
010001	0010	100110	1100	111011	XXXX
010010	0101	100111	1111	111100	XXXX
010011	0111	101000	0001	111101	XXXX
010100	0111	101001	0011	111110	XXXX
				111111	XXXX

TABLE T13

ADDRESS	CONT	ADDRESS	CONT	ADDRESS	CONT
000000	0000	010101	0001	101010	1111
000001	0011	010110	0100	101011	0010
000010	0110	010111	0111	101100	1101
000011	1001	011000	1111	101101	0000
000100	0100	011001	0010	101110	0011
000101	0111	011010	0101	101111	0110
000110	1010	011011	1000	110000	1110
000111	1101	011100	0011	110001	0001
001000	0101	011101	0110	110010	0100
001001	1000	011110	1001	110011	0111
001010	1011	011111	1100	110100	0010
001011	1110	100000	0100	110101	0101
001100	1001	100001	0111	110110	1000
001101	1100	100010	1010	110111	1011
001110	1111	100011	1101	111000	XXXX
001111	0010	100100	1000	111001	XXXX
010000	1010	100101	1011	111010	XXXX
010001	1101	100110	1110	111011	XXXX
010010	0000	100111	0001	111100	XXXX
010011	0011	101000	1001	111101	XXXX
010100	1110	101001	1100	111110	XXXX
				111111	XXXX

TABLE T14

ADDRESS	CONT	ADDRESS	CONT	ADDRESS	CONT
000000	0000	010101	0011	101010	0110
000001	0001	010110	0100	101011	0111
000010	0010	010111	0101	101100	0101
000011	0011	011000	0011	101101	0110
000100	0001	011001	0100	101110	0111
000101	0010	011010	0101	101111	1000
000110	0011	011011	0110	110000	0011
000111	0100	011100	0100	110001	0100
001000	0010	011101	0101	110010	0101
001001	0011	011110	0110	110011	0110
001010	0100	011111	0111	110100	0100
001011	0101	100000	0010	110101	0101
001100	0011	100001	0011	110110	0110
001101	0100	100010	0100	110111	0111
001110	0101	100011	0101	111000	0101
001111	0110	100100	0011	111001	0110
010000	0001	100101	0100	111010	0111
010001	0010	100110	0101	111011	1000
010010	0011	100111	0110	111100	0110
010011	0100	101000	0100	111101	0111
010100	0010	101001	0101	111110	1000
				111111	1001

TABLE T15

IV. PUBLICATIONS

During the past grant-year the following publications have appeared on the work supported by this grant :

(1) Alan Huang, Yoshito Tsunoda, J. W. Goodman, and Satoshi Ishihara, "Optical Computation Using Residue Arithmetic", Applied Optics, Vol. 18, No. 2, January 1979, pp. 149-162

(2) Alan Huang, Yoshito Tsunoda, J. W. Goodman, and Satoshi Ishihara, "Some Optical Methods for Performing Residue Arithmetic Operations", Proceedings of the 1978 International Optical Computing Conference, Sept. 5-7, 1978, London, England. (IEEE 78CH1305-2C)

(3) Alan Huang, "An Optical Residue Arithmetic Unit", 5th Annual Symposium on Computer Architecture, Conference Proceedings, Institute of Electrical and Electronic Engineers (78CH1284-9C), April 1978

AFOSR Contractors and Grantees

Dr David Casasent
Carnegie-Mellon University
Department of Electrical Engineering
Pittsburgh, Pennsylvania 15213

Dr B. Jin Chang
Radar and Optics Division
Environmental Research Institute of Michigan
P. O. Box 618
Ann Arbor, Michigan 48107

Dr George Eichmann
Department of Electrical Engineering
The City University of New York
Covent Avenue at 138th Street
New York, N.Y. 10031

Dr Elsa Garmire
Center for Laser Studies
University of Southern California
Los Angeles, California 90007

Dr Nicholas George
Director, Institute of Optics
The University of Rochester
Rochester, New York 14627

Dr Joseph W. Goodman
Department of Electrical Engineering
Stanford Electronics Laboratories
Stanford University
Stanford, California 94305

Dr Bobby R. Hunt
Systems & Industrial Engineering Dept
University of Arizona
Tucson, Arizona 85721

Mr Peter Kellman
ESL Incorporated
495 Java Drive
Sunnyvale, California 94086

Dr Sing H. Lee
Dept of Applied Physics and Information Science
University of California, San Diego
La Jolla, California 92093

Mr Kenneth Leib
Research Department
Grumman Aerospace Corporation
South Oyster Bay Road
Bethpage, New York 11714

Prof Emmett N. Leith
Electrical and Computer Engineering Dept
The University of Michigan
Ann Arbor, Michigan 48109

Dr William T. Rhodes
School of Electrical Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

Dr Alexander A. Sawchuk
Electrical Engineering Dept
University of Southern California
Los Angeles, California 90007

Mr Bernard Soffer
Opto-Electronics Department
Hughes Research Laboratories
3011 Malibu Canyon Road
Malibu, California 90265

Dr William Steier
Co-Chairman, Electrical Engineering Dept
University of Southern California
Los Angeles, California 90007

Dr C. S. Tsai
Department of Electrical Engineering
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Dr John Walkup
Department of Electrical Engineering
Texas Tech University
Lubbock, Texas 79409

Dr Cardinal Warde
Electrical Engineering & Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Supplemental Distribution List

Dr Gerald Brandt
Westinghouse Research & Development Center
1310 Beulah Road
Pittsburgh, Pennsylvania 15235

Dr Keith Bromley
Naval Ocean Systems Center
Code 8111
271 Catalina Blvd
San Diego, California 92152

Dr Robert Brooks
TRW Systems Group
R1/1062 One Space Park
Redondo Beach, California 90278

Dr F. Paul Carlson
Applied Physics and Electronic Science
Oregon Graduate Center
19600 N.W. Walker Road
Beaverton, Oregon 97005

Prof W. Thomas Cathey
Dept of Electrical Engineering
University of Colorado
Denver, Colorado 80302

Dr H. J. Caulfield
Aerodyne Research, Inc
Applied Science Division
Bedford Research Park
Bedford, Massachusetts 01730

Dr Edwin Champagne
AFAL/DH
Wright-Patterson AFB, Ohio 45433

Prof Stuart A. Collins, Jr
Dept of Electrical Engineering
Ohio State University
2015 Neil Avenue
Columbus, Ohio 43210

Dr Nabil Farhat
University of Pennsylvania
200 South 33rd Street
Philadelphia, Pennsylvania 19174

Dr David Flannery
Mr Mike Hamilton
AFAL/DHO
Wright-Patterson AFB, Ohio 45433

Dr Albert Friesem
Weizmann Institute of Science
Rehovot, Israel

Dr Bobby Guenther
Commander
U.S. Army Missile Res. and Dev. Command
Attn: DRDMI - TRO/B. D. Guenther
Redstone Arsenal, Alabama 35809

Mr Peter S. Guilfoyle
Department 232, Mail Station 13-3
McDonnell Douglas Astronautics Company
5301 Bolsa Avenue
Huntington Beach, California 92647

Dr Richard Hudgins
Itek Corporation
10 Maguire Road
Lexington, Massachusetts 02173

Dr T. C. Lee
Corporate Research Center
Honeywell, Inc.
10701 Lyndale Ave., So.
Bloomington, Minnesota 55420

Prof Adolf Lohmann
Physics Institute
University of Erlangen-Nurnberg
Erwin-Rommel-Strasse
D 8520 Erlangen
F.R. Germany

Mr Bob V. Markevitch
Ampex Corporation
401 Broadway
Redwood City, California 94063

Dr Robert Marks
Dept of Electrical Engineering
University of Washington
Seattle, Washington 98195

Captain Bill Miceli
RADC/ESO
L. G. Hanscom AFB, Massachusetts 01730

Dr Robert A. Sprague
Xerox Corporation
Palo Alto Research Labs
3333 Coyote Hill Road
Palo Alto, California 94304

Mr Eric Stevens
Code 7924S
Naval Research Laboratory
Washington, D.C. 20375

Dr William Stoner
Systems Applications, Inc.
3 Preston Court
Bedford, Massachusetts 01730

Dr Henry Taylor
Rockwell International Science Center
Thousand Oaks, California 91360

Prof Brian Thompson, Dean
School of Engineering
University of Rochester
Rochester, New York 14627

Mr Terry Turpin
NSA R551
9800 Savage Road
Fort Meade, Maryland 20755

Dr Anthony Vander Lugt
232 Cocoa Avenue
Indialantic, Florida 32903

Dr Bernard Vatz, Radar Directorate
BMDATC
P. O. Box 1500
Huntsville, Alabama 35807

Dr Carl M. Verber
Battelle Columbus Laboratories
505 King Avenue
Columbus, Ohio 43201

Mr Harper Whitehouse
Naval Ocean Systems Center
San Diego, California 92152

Prof James Wyant
Optical Sciences Center
University of Arizona
Tucson, Arizona 85721

Dr Francis T. S. Yu
Electrical and Computer Engineering
Wayne State University
Detroit, Michigan 48202